NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

USING NEGATIVE INFORMATION TO IMPROVE PERFORMANCE OF FORWARD SCATTER ARRAYS

by

Daniel B. Widdis

March, 1995

Thesis Advisor: Alan R. Washburn

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995.	Master's Thesis

4. TITLE AND SUBTITLE USING NEGATIVE INFORMATION TO IMPROVE PERFORMANCE OF FORWARD SCATTER ARRAYS	5. FUNDING NUMBERS
6. AUTHOR(S) Widdis, Daniel B.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited.	

13. ABSTRACT *(maximum 200 words)*

Many tracking algorithms, such as implementations of Kalman filters, use only target positioning data as input. They ignore negative information from sensors that do not detect the target. Recent improvements in computing performance allow the development of tracking algorithms that can fuse information from many sources, including negative information, into the target motion analysis. This thesis evaluates the significance of negative information in a discrete tracking algorithm applied to a tracking scenario in which an array of forward scatter tripwire sensors covers the search area. Additionally, this thesis explores the effect of selected arrangements of an array of tripwire sensors and performance parameters on tracking capability. Using negative information significantly improves tracking performance, especially in a cost-effective arrangement of tripwires where several lines of position are coincident.

14. SUBJECT TERMS FORWARD SCATTER TRIPWIRE NEGATIVE INFORMATION DISCRETE TRACKING TARGET MOTION ANALYSIS	15. NUMBER OF PAGES 89
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

USING NEGATIVE INFORMATION TO IMPROVE PERFORMANCE OF FORWARD SCATTER ARRAYS

by

Daniel B. Widdis
Lieutenant, United States Navy
B.S., United States Naval Academy, 1988

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1995

Author: _____
Daniel B. Widdis

Approved by: _____
Alan R. Washburn, Thesis Advisor

_____
Robert F. Dell, Second Reader

_____
Peter Purdue, Chairman
Department of Operations Research

ABSTRACT

Many tracking algorithms, such as implementations of Kalman filters, use only target positioning data as input. They ignore negative information from sensors that do not detect the target. Recent improvements in computing performance allow the development of tracking algorithms that can fuse information from many sources, including negative information, into the target motion analysis. This thesis evaluates the significance of negative information in a discrete tracking algorithm applied to a tracking scenario in which an array of forward scatter tripwire sensors covers the search area. Additionally, this thesis explores the effect of selected arrangements of an array of tripwire sensors and performance parameters on tracking capability. Using negative information significantly improves tracking performance, especially in a cost-effective arrangement of tripwires where several lines of position are coincident.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Many tracking algorithms or trackers operate using only sensor information that indicates a potential target. They thus ignore the negative information from sensors that do not detect the target. These trackers, such as the Maneuvering Target Statistical Tracker (MTST), are frequently based on extended Kalman filters (Stone, 1991).

Recent improvements in computing performance allow the development of trackers that can fuse information from many sources, including negative information, into the target motion analysis. The primary examples are search tactical decision aids (TDAs) such as VPCAS, PACSEARCH, ASWTDA, (Wagner, 1989), and Nodestar (Stone, 1991). There is sufficient computational difficulty in the fusion of negative information to warrant investigation of the significance of this data, and evaluation of the accuracy of the resulting target distributions.

This thesis considers a discrete tracking algorithm applied to a tracking scenario in which an array of forward scatter *tripwires* covers the search area. The tripwire model, based on current research at the Johns Hopkins University Applied Physics Laboratory (JHU-APL) in bistatic forward scatter acoustic arrays, provides a long-range line segment capable of reporting, with certain probability, the presence (or absence) of a target crossing the tripwire during a given time interval. Practical considerations determine a few basic arrangements of an array of such tripwires.

Some mathematical models of the performance of such an array assume that the contribution of negative information to the tracking algorithm is negligible (Loane, September 1993, p. 9). This thesis shows that fusion of negative information into a tracking algorithm can significantly enhance its performance. The effect is most distinct in arrangements of line-of-position sensors that share a common point, and in situations where spurious (false) alarms produce a large number of detections. In the case of forward scatter arrays, where cost-effective arrangements of tripwires will likely have coincident endpoints, fusion of negative information into the target motion analysis is necessary to produce accurate results.

Implementation of tracking using negative information requires estimates of the probabilities of target detection as well as false alarms. While improper assumptions degrade tracker performance, their effect is less significant than choosing not to take negative information into account.

In a source-receiver implementation of tripwires, practical implementation of an array of tripwires would involve placement of many receivers around each source. Constant-cost analysis of the effects of leaving gaps in, or overlapping, coverage, shows that arrangement of the sources just far enough apart to provide complete coverage results in the best localization of the target.

Addition of tripwires can continually enhance the performance of a field of tripwires. This thesis indicates that a minimum number of tripwires are needed to provide adequate tracking capability, with further additions marginally improving performance.

This thesis may be used as a starting point for further analysis into the significance of negative information for other types of tracking systems, and provides useful insight into the practical arrangement of a forward scatter tripwire array.

I. INTRODUCTION

A. PROBLEM DEFINITION

Many tracking algorithms or trackers operate using only sensor information that indicates a potential target. They thus ignore the negative information from sensors that do not detect the target. These trackers, such as the Maneuvering Target Statistical Tracker (MTST), are frequently based on extended Kalman filters (Stone, 1991).

Recent improvements in computing performance allow the development of trackers that can fuse information from many sources, including negative information, into the target motion analysis. The primary examples are search tactical decision aids (TDAs) such as VPCAS, PACSEARCH, ASWTDA, (Wagner, 1989), and Nodestar (Stone, 1991). There is sufficient computational difficulty in the fusion of negative information to warrant investigation of the significance of this data, and evaluation of the accuracy of the resulting target distributions.

This thesis considers a discrete tracking algorithm applied to a tracking scenario in which an array of forward scatter tripwires covers the search area. The tripwire model, based on current research at the Johns Hopkins University Applied Physics Laboratory (JHU-APL) in bistatic forward scatter acoustic arrays, provides a long-range line segment capable of reporting, with certain probability, the presence (or absence) of a target crossing the tripwire during a given time interval. Practical considerations determine a few basic arrangements of an array of such tripwires.

Some mathematical models of the performance of such an array assume that the contribution of negative information to the tracking algorithm is negligible (Loane, September 1993, p. 9). This thesis evaluates the validity of that assumption, as well as the merits of selected tripwire array arrangements.

B. TRACKING ALGORITHM AND SIMULATION

This thesis develops a simulation and tracking algorithm, called *HexTrack*, programmed in Turbo Pascal (Borland, 1992). *HexTrack* incorporates detection

information from a simulated array of tripwire sensors arranged in a search area. *HexTrack* generates *real* detections by simulating target motion in a discrete target position state space, although the tracking algorithm does not use its knowledge of actual target location. *HexTrack* also generates spurious detections (*false alarms*) from the tripwire sensors. Two separate trackers generate likelihood distributions: the *positive information tracker* incorporates only detection information, and the *negative information tracker* incorporates both alarm and non-alarm information. *HexTrack* then compares the likelihood distributions to the simulated target position to evaluate tracker performance.

The target state space consists of 6,290 hexagonal cells covering a 150 by 150 nautical mile search area. *HexTrack* uses hexagonal cells to take advantage of radial symmetry for target motion and to avoid the complications of cells adjacent to each other only at corners. (Six other cells border each interior cell on its edges.) Memory, data segment, and heap size constraints in Turbo Pascal limit the number of cells. *HexTrack* calculates an alarm probability for each cell-tripwire combination, accounting for the probability that the tripwire alarm is a result of target presence in that cell, as well as the probability of a false alarm.

HexTrack initializes the trackers with likelihood distributions uniform over the search area and the target in a randomly chosen cell. On each iteration, representing a step of discrete time, *HexTrack*:

1. Updates actual target position, using a random number generator.

2. Calculates new likelihood distributions, based on the likelihood distribution at the end of the previous iteration and target motion probabilities (motion update).

3. Calculates tripwires alarming, using a random number generator and conditioning on actual target position.

4. Updates both trackers' likelihood distributions to incorporate tripwire alarm information (positive information update), if any.

5. Updates the negative information tracker's likelihood distribution to incorporate tripwire non-alarm information (negative information update).

6. Calculates and records tracker performance statistics.

C. MEASURES OF EFFECTIVENESS

Point estimates of target position are the basis for the usual measure of effectiveness (MOE) for a tracking system, missed distance. Since *HexTrack* produces likelihood distributions of target position, a more appropriate MOE is mean missed distance. (Stone, 1991, p. 8) Additionally, this thesis uses an MOE representing accuracy of the likelihood distributions. Chapter III discusses these MOEs in detail.

II. TRACKING MODEL

A. TRIPWIRE DETECTION MODEL

The simulated sensors providing detection/non-detection information to *HexTrack* are based on bistatic forward scatter acoustic arrays. These arrays operate by transmitting sound omnidirectionally from an acoustic source. When a target (normally a submarine) is near the line between the source and a remote receiver, the target scatters additional sound energy towards the receiver. The additional energy at the receiver, if above a predetermined threshold, indicates a detection. (Loane, December 1993) Because the target scatters the most acoustic energy directly opposite the source (forward scattering), detections only occur in the vicinity of the line between the source and receiver, hence the term *tripwire*. Figure 1 illustrates the basic operation of a forward scatter array.

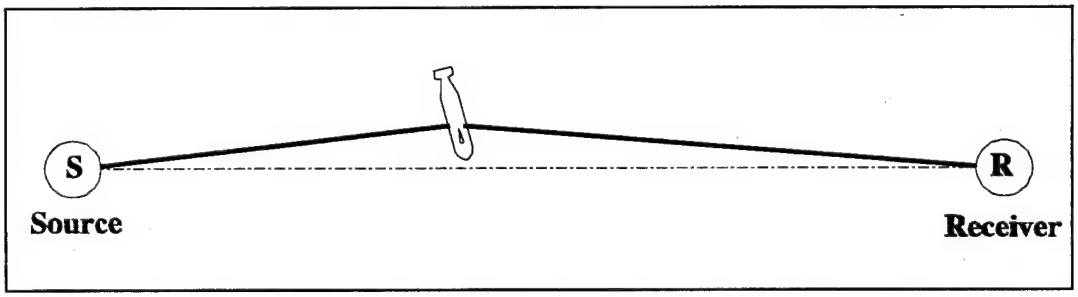

Figure 1. Operation of a forward scatter array (tripwire). As the target passes between the source and receiver, sound from the source is scattered forward past the target. The additional noise at the receiver is interpreted as a detection.

Forward scatter arrays are capable of detection ranges much longer than other bistatic arrays (Loane, 1992). A realistic, unclassified estimate of maximum detection range is 50 miles. This thesis models tripwires as rectangular areas two miles wide and up to 50 miles in length. Detections occur as Bernoulli trials, occurring once for each tripwire at the end of 7.06 minute time intervals. The target motion model is the basis for interval length. *HexTrack* assumes the probability that a tripwire detects the target during a single time interval is uniform over the entire search area, independent from interval to interval, and independent of other tripwires. In addition to detections caused

by target presence in the detection area, each tripwire has a probability during each time interval of generating a spurious (false) alarm, independent of other tripwires, the target, and from interval to interval. In reality, detection and false alarm probabilities would vary with geographic and environmental conditions, and potentially be subject to interference from the acoustic sources of other tripwires.

B. TRIPWIRE FIELD PATTERNS

Each receiver can potentially detect along the line to multiple sources. Cost considerations encourage use of this feature. Since acoustic sources are at least an order of magnitude more expensive than receivers (Loane, 1992), a field of tripwires would likely consist of multiple receivers arranged around each source. Figure 2 shows an example of a portion of such a field. Appendices A, B, and C show this *Spoke* pattern and other tripwire patterns considered in this thesis.

C. TARGET MOTION MODEL

Target motion in *HexTrack* is based on a Markov state transition matrix operating on target position at each iteration. During each iteration, the target remains in the same cell with some probability, or moves to one of the six adjacent cells.

A separate simulation allowed empirical determination of position state transition probabilities, both unconditional and conditioned on the position state transition on the previous iteration, which represents a velocity state. Appendix D lists the target motion assumptions, simulation, and resulting transition probabilities. The motion update for the likelihood distribution does not use target velocity information due to programming limitations on numbers of variables. Chapter IV explores the significance of this simplification.

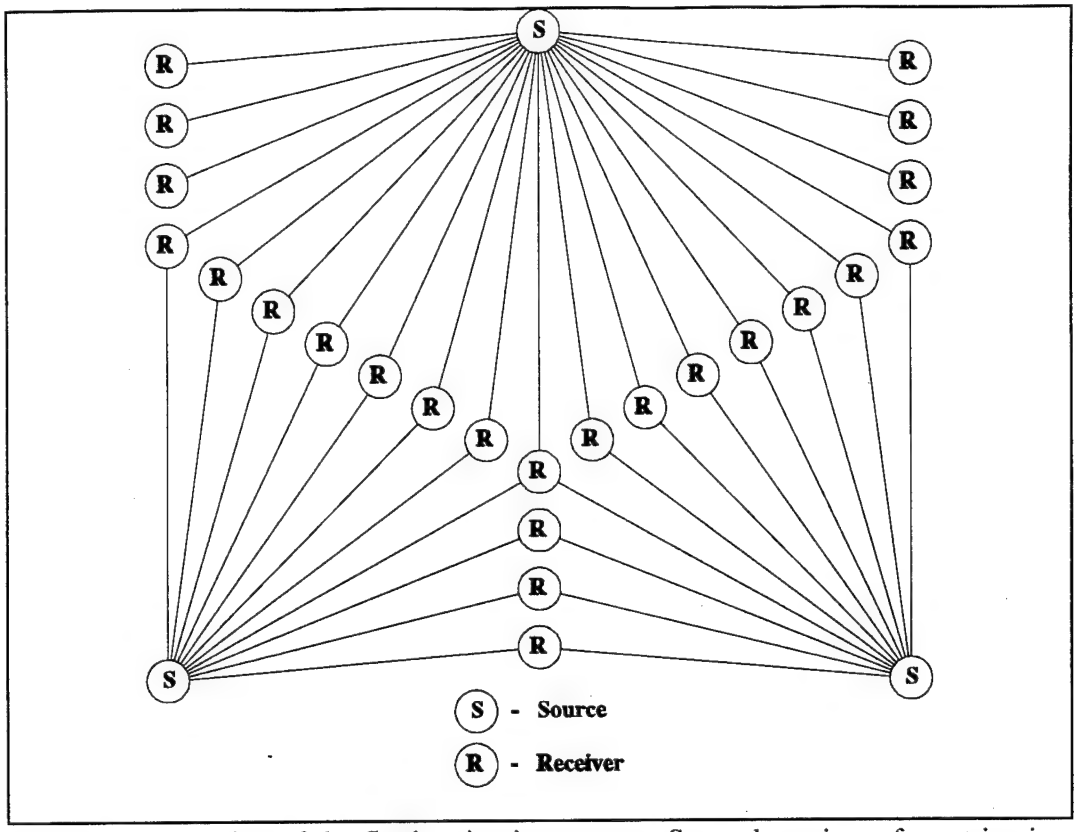

Figure 2. A portion of the *Spoke* tripwire pattern. Several receivers form tripwires with each source. Only a repeating element of the pattern is illustrated; the sources are arranged in equilateral triangles to fill up space.

D. FUSION OF DETECTION AND NON-DETECTION INFORMATION

HexTrack performs three major steps each iteration: a motion update, an information update, and display of the updated distribution. Appendix E lists the *HexTrack* source code. The following sections outline the computations performed in each step.

1. Motion Update

The tracker assumes target motion is determined by a Markov transition probability matrix M, with entries

$$M_{ij} = \Pr(\text{target in cell } i \text{ moves to cell } j),$$

where row sums

$$\sum_j M_{ij} = 1 \ . \tag{1}$$

Actual target motion in the *HexTrack* simulation is determined by the same matrix M in cases where the velocity state is ignored.

Let Y_k be the target's position at iteration k. Let P_k be a row vector of probabilities

$$P_{ik} = \Pr(Y_k=i \mid \text{information through iteration } k),$$

where

$$\sum_i P_{ik} = 1 \ . \tag{2}$$

P_k is the tracker's presumed distribution of target position at the end of the kth iteration, given the initial distribution and all motion and information updates from k and previous iterations.

The motion update from iteration k to $k+1$ is

$$P^-_{k+1} = P_k M \ , \tag{3}$$

where

$$P^-_{i,k+1} = \Pr(Y_{k+1}=i \mid \text{information through iteration } k).$$

The superscript - denotes that this distribution does not yet include information from iteration $k+1$.

2. Information Update

Tripwires alarm (report a detection) either as a result of target presence in a cell near the detecting tripwire or as a result of a spurious alarm. For each cell i and tripwire j, calculate an alarm probability

$$A_{ij} = \Pr(\text{tripwire } j \text{ alarms given target is in cell } i).$$

For each cell i, define the non-alarm probability NAP_i as the probability that no tripwires alarm if the target is in that cell. Then

$$NAP_i = \prod_j (1-A_{ij}). \tag{4}$$

For the *positive information tracker*, the information update for each cell i is

$$P^{+}_{i,k+1} = P^{-}_{i,k+1}\left[\prod_{\substack{j \\ alarmed}} A_{ij}\right] . \tag{5}$$

The *negative information tracker* includes the probability of tripwires not alarming in the information update:

$$P^{+}_{i,k+1} = P^{-}_{i,k+1}\left[\prod_{\substack{j \\ alarmed}} A_{ij}\right]\left[\prod_{\substack{j\ not \\ alarmed}} (1 - A_{ij})\right] . \tag{6}$$

Note that the *negative information tracker* uses both positive (detection) and negative (non-detection) information. The abbreviation of the description is made for ease of readability.

For ease of calculation, Equation 6 can be rewritten

$$P^{+}_{i,k+1} = P^{-}_{i,k+1}\left[\prod_{\substack{j \\ alarmed}} \frac{A_{ij}}{(1 - A_{ij})}\right] NAP_i . \tag{7}$$

The distribution is normalized to a probability distribution by

$$P_{i,k+1} = \frac{P^{+}_{i,k+1}}{\sum_i P^{+}_{i,k+1}} . \tag{8}$$

Equations 6 and 8 are a straightforward application of Bayes' formula (Ross, 1993, p. 14). The use of products of probability reflects the assumption of independence of detections.

3. Display

After each iteration, *HexTrack* displays the likelihood distributions. *HexTrack* sorts cells by P_{ik} and displays them with brighter colors representing higher likelihoods.

Each color change represents the boundary of an Area of Uncertainty (AOU) or containment region. The display also shows actual target location, permitting qualitative analysis of tracker performance.

Figure 3 is an example of *HexTrack*'s display. The color shades are reversed such that the darker shades of grey represent the brighter colors on the display and indicate higher likelihood. The actual target position, unknown to the tracker, appears as a white (black in the figure) dot just above and to the left of the center of the search area. The target is near a tripwire that it caused to alarm; false alarms have occurred on several other tripwires in the search area. The likelihood distribution is higher for both trackers on the alarming tripwires, highest where the alarmed tripwires are near enough that target motion could account for both alarms. The negative information tracker shows the effect of non-alarm information, suppressing the likelihood distribution on tripwires that did not alarm. Most of the likelihood for the positive information tracker is in the center, where six alarmed tripwires intersect.

Note that Figure 3 is after only two iterations, so very little information has been accumulated for either tracker, and the distribution is highly multimodal. Figures 6 and 7, in Chapter IV, show similar comparisons after several more iterations.

The statistics shown at the bottom of Figure 3 are examples of the performance statistics *HexTrack* computes during the display process. Chapter III discusses the purpose and calculation of these statistics.

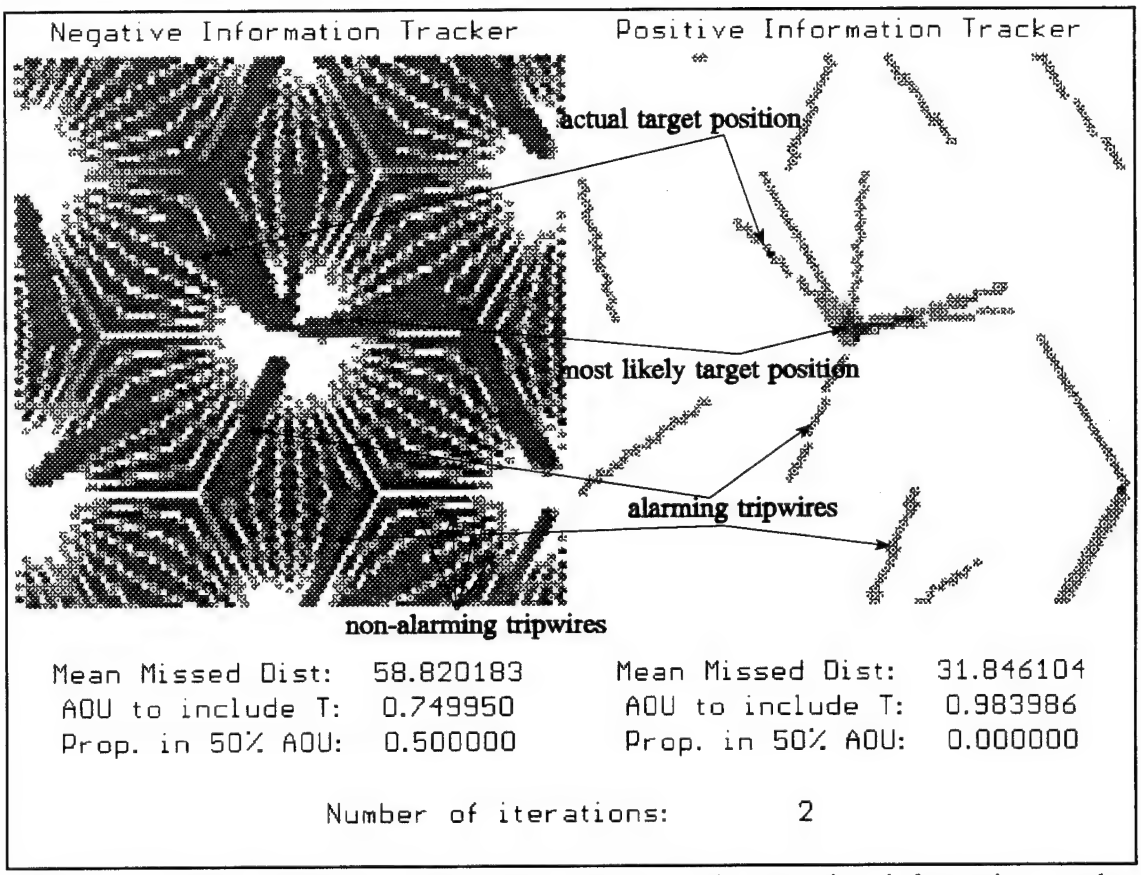

Figure 3. Screen capture of *HexTrack* display. The negative information tracker suppresses likelihood at tripwires that do not alarm. The positive information tracker's distribution is highest at the intersection of the alarmed tripwires.

III. DATA ANALYSIS METHODS

To quantitatively compare the performance of the negative information tracker and the positive information tracker, appropriate Measures of Effectiveness (MOEs) are necessary. The MOEs used relate to two factors: the tracker's ability to estimate true target position, and the accuracy of the calculated distribution. The distances from target locations predicted by the tracker to the actual target location, weighted by likelihood, represent the ability of the tracker to estimate target position. With this measure, smaller is better. Accuracy is a statistical measure of how well the target likelihood distribution represents the tracker's uncertainty of the target's position. Accuracy measures range from 0 to 100%, where larger is better. The following sections formally define these MOEs.

Both MOEs discussed rely on knowledge of *actual* target position for calculations. Although *HexTrack* keeps track of target position for the generation of real sensor contacts and collection of these performance statistics, the trackers do not use this knowledge to generate the likelihood distributions.

HexTrack records data for 50 iterations, approximating a six hour tracking period, replicating each set of parameters 60 times. At the beginning of each replication, *HexTrack* resets the target likelihood distributions to uniform over the search area, and relocates the target randomly in the search area. Both MOEs are then applied to the resulting 3000 data points.

A. MEAN MISSED DISTANCE (MMD)

The usual measure of tracker performance for trackers that produce point estimates of target position is missed distance. This measure is not appropriate for *HexTrack* because *HexTrack* does not forecast a specific target position, but a generalization of this measure is straightforward. Let

p_i = calculated probability the target is in cell i

and

d_i = distance from cell i to the target position.

Then the root mean squared missed distance (Stone, 1991, p. 10) is

$$MMD = \sqrt{\sum_i p_i d_i^2}. \tag{9}$$

HexTrack calculates MMD on every iteration of the simulation, and uses the mean MMD over all iterations and replications.

B. ACCURACY

Using mean missed distance as the sole MOE is not sufficient. A specific example is a bimodal target distribution. A unimodal distribution with a peak between the modes of the actual distribution could have a lower MMD than the actual distribution. In the MMD calculation, it is better to be half-right all of the time than to be right only half the time. Figure 4 shows a one-dimensional example of this problem. The bimodal distribution has a missed distance of 0 with probability 0.5 and 2 with probability 0.5, resulting in an MMD of 1.414. The unimodal distribution has a missed distance of 1 with probability 1.0. An MOE representing the accuracy of a tracker's distribution is necessary.

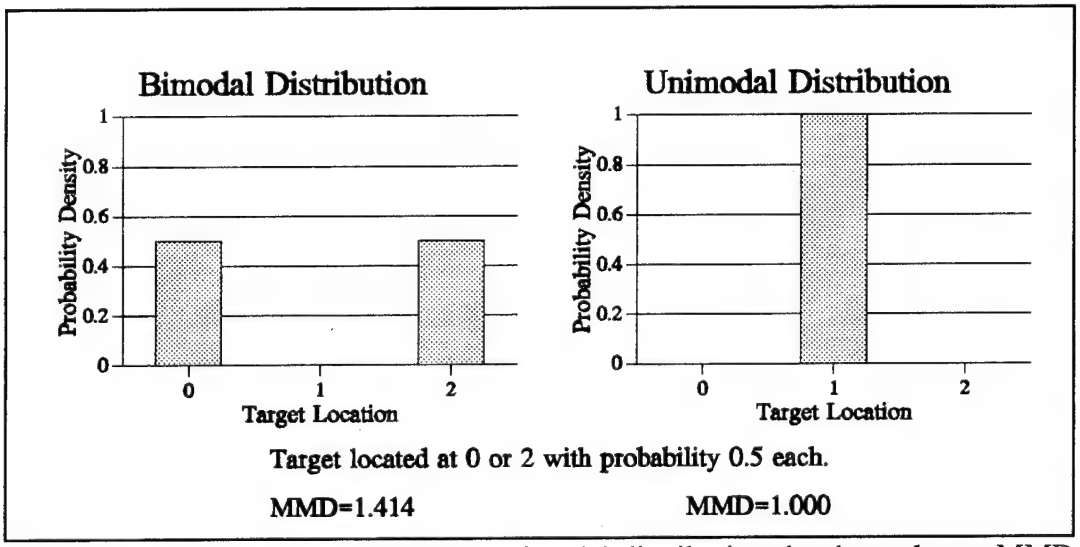

Figure 4. Example of an inaccurate unimodal distribution that has a lower MMD than the actual bimodal distribution. An MOE measuring accuracy is necessary.

The target position probability distribution allows calculation of Areas of Uncertainty (AOUs). Trackers based on Kalman filters, whose distribution for target position is bivariate normal, frequently specify a 2σ (86.5%) uncertainty ellipse around the mean estimated target position. If the tracker is accurate, this AOU contains the target 86.5% of the time. *HexTrack* produces analogous containment regions by sorting the probabilities that the target is in each cell from highest to lowest. The X% containment region consists of the highest likelihood cells that must be summed to reach X% cumulative probability.

On each iteration, *HexTrack* records the smallest containment region that includes the target. This region includes all cells with likelihood higher than the cell containing the target and a random portion of the cell containing the target. It is the discrete counterpart of the proportion of a bivariate normal distribution contained inside an ellipse intersecting the target. The containment percentile of this region, for a target in cell t on iteration k, is

$$x_k = \sum_{i|p_i>p_t} p_i + u(p_t) , \tag{10}$$

where u is a uniform random variable from [0,1]. If the tracker is accurate, the x_k values correspond to selection of random variables from a uniform distribution on [0,1]. The multiplication by u in Equation 10 assures that x_k is uniformly distributed, as long as the target's location i actually has the distribution p_i. (Stone, 1991, p. 9)

To calculate accuracy, all x_k, from 60 replications of 50 iterations each, are sorted with $x_{(1)}$ being the smallest and $x_{(3000)}$ the largest. These values define an empirical distribution function:

$$F(x) = \begin{cases} 0 & \text{for } x < x_{(1)} \\ \dfrac{k}{3000} & \text{for } x_{(k)} \le x < x_{(k+1)} , \quad k=1,2,\ldots,2999 \\ 1 & \text{for } x \ge x_{(3000)} . \end{cases} \tag{11}$$

The Kolmogorov-Smirnov (K-S) statistic, representing the maximum deviation of this empirical function from the uniform distribution, is

$$D = \max_{x} |F(x) - x| \ . \tag{12}$$

Accuracy is then defined (Stone, 1991, p. 12) as

$$A = 100(1 - D)\ \% \ . \tag{13}$$

If the tracker produces a likelihood distribution that accurately represents its uncertainty in estimating target position, D is near zero and Accuracy is near 100%. Note that D is a K-S statistic in spite of the fact that the distribution of the target's position is discrete because of the inclusion of a random portion of the cell containing the target.

Figure 5 shows an example of accuracy measurements. In the pessimistic tracker, the worst error shows the 42% containment region containing the target 79% of the time. The optimistic tracker contains the target in the 42% containment region only 16% of the time. The accurate tracker contains the target in the 19% containment region 22% of the time.

It is important to observe that Accuracy alone is not a sufficient MOE. A likelihood distribution that continues to assume the target is distributed uniformly over the search area produces a very accurate distribution, but with a high MMD. Roughly speaking, the best tracker minimizes MMD among trackers that are highly accurate.

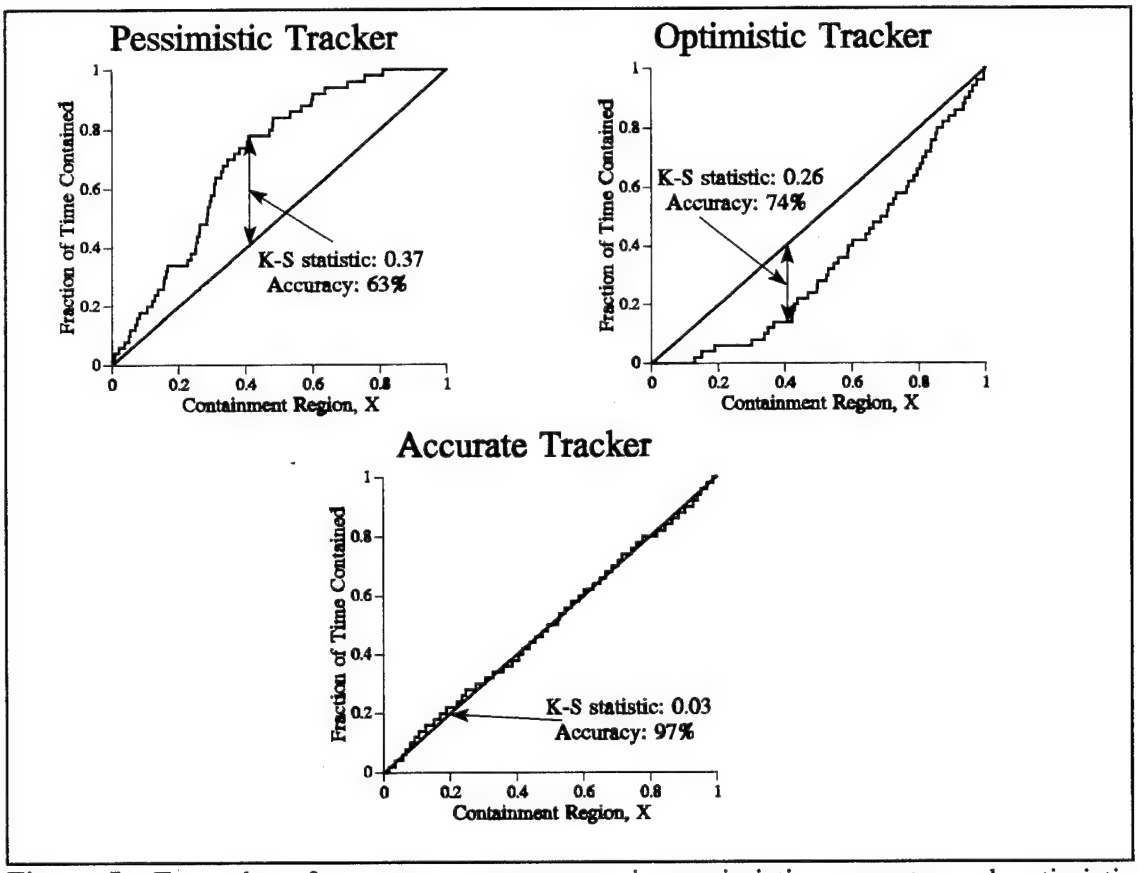

Figure 5. Examples of accuracy measurement in pessimistic, accurate, and optimistic trackers. Both pessimistic and optimistic trackers result in low accuracy.

IV. RESULTS

This chapter considers three different patterns of tripwire arrangement. In addition to a *spoke* pattern that uses multiple receivers arranged around each source, a regular square *grid* of tripwires and (uniform) *random* placement of tripwires are considered. Appendix A shows these three arrangements, which are comparable in the cumulative *length* of tripwires in the search area. Since the tripwire model assumes the sensors cover the area of their length and a fixed width, this comparison of patterns considers comparable *density* of tripwire coverage. Cost considerations are deferred until Chapter V.

A. NEGATIVE INFORMATION SIGNIFICANCE

A tracker that uses only positive information is at a great disadvantage when given only line-of-bearing information. When several lines intersect at a common point, such as the source location in the spoke arrangement, the target distribution becomes artificially high near the source. Additionally, the tracker allows the distribution to expand into regions near the source even during periods in which no tripwires alarm in that region. The comparison outlined in this section highlights this disadvantage.

Observation of the tracker displays during the simulation runs shows that the tracker using only positive information generates a distribution with highest likelihood at and around the intersection of alarming tripwires. The tracker incorporating negative information suppresses the target distribution at these intersections when the tripwires do not alarm, and the likelihood distribution is highest at alarming tripwires and in the areas between non-alarming tripwires.

Figure 6 shows an example of typical tracker behavior when tripwires are arranged in the spoke pattern. In this example, only two tripwires have alarmed. The positive information tracker's distribution is higher near the intersection of the two tripwires, at the centrally located source. The negative information tracker considers the fact that several other tripwires in that area did *not* alarm and suppresses the distribution there,

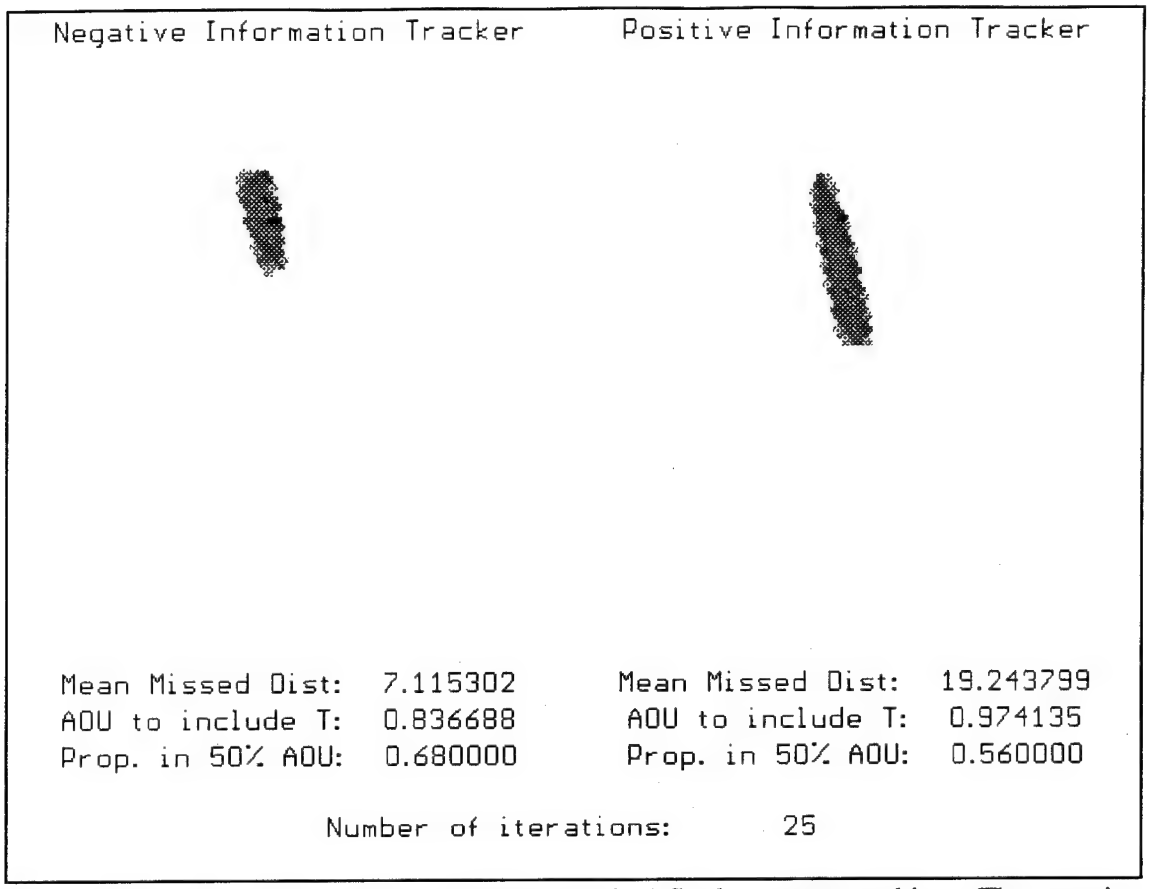

Figure 6. *HexTrack* display demonstrating typical *Spoke* pattern tracking. The negative information tracker suppresses the distribution where other tripwires have not alarmed, resulting in better localization of the target.

causing the distribution to be properly concentrated much nearer the target. The effect shown in Figure 6 becomes even more pronounced as additional tripwires alarm, especially if the alarms are false. Typically the positive information tracker's distribution in these circumstances is almost exclusively in cells immediately surrounding the source common to the highest number of alarming tripwires.

Table I summarizes Accuracy and Mean Missed Distance statistics for the positive information tracker, that used only detection information, and the negative information tracker, that incorporated both detection and non-detection information, for the three tripwire arrangements considered, at three levels of false alarm probability.

False Alarm Prob.	Tripwire Arrangement	Average MMD (miles)		Accuracy (%)	
		Negative	Positive	Negative	Positive
0.00	Spoke	11.85*	20.05	98.80*	54.57
	Grid	8.76	8.58*	98.97*	81.17
	Random	19.59*	20.84	96.14*	86.55
0.01	Spoke	21.41*	37.30	96.91*	25.32
	Grid	18.43	17.91*	97.07*	81.11
	Random	28.78*	31.04	97.08*	61.82
0.10	Spoke	45.22*	63.05	97.77*	5.11
	Grid	37.30*	38.91	97.71*	83.19
	Random	45.78*	54.94	97.28*	35.12

* denotes winner

Table I. Tracker performance statistics for the basic target motion model. The Accuracy of the *negative information tracker* is better than the *positive information tracker* in every case. The *positive information tracker* has a slightly lower MMD in only two cases, using the grid pattern with zero or low levels of false alarms.

The accuracy statistics for the negative information tracker are all above 96%. This is an expected result of these cases, in which the assumptions exactly match actual target motion. Accuracy can be expected to asymptotically approach 100% as these cases are repeated.

Accuracy of the positive information tracker shows extremely significant degradation caused by the intersection of the lines of position. The worst performance is with the spoke arrangement, where 42 tripwires intersect at the location of the source. The magnitude of the error increases at higher rates of false alarms. The grid arrangement is the best for the *positive information tracker*, as only two tripwires intersect at any location in this pattern. The accuracy is still significantly worse than with the negative information tracker.

The MMD statistics reflect the effect of these inaccurate distributions. In the spoke arrangement, the negative information tracker is significantly better than the positive information tracker. The negative information tracker shows slightly better performance in the random arrangement, increasing as the false alarm rate increases, causing accuracy of the positive information tracker to decrease. The only cases in which the positive information tracker has a lower MMD than the negative information tracker are the grid arrangement, with zero or low levels of false alarms. This apparent improvement, however, is small, and is a result of the inaccurate distribution maintained by the positive information tracker. Observation of *HexTrack* during these simulation runs shows that the negative information tracker frequently produces a bimodal likelihood distribution, centered around the last tripwire to detect the target, with the target in one of the two peaks. The positive information tracker does not suppress the distribution on the non-detecting tripwire, centering the likelihood distribution between the two most likely target positions. As shown in Figure 4 of Chapter III, this averaging effect can result in a lower MMD.

Figure 7 shows an example of typical grid arrangement likelihood distributions. The negative information tracker's distribution is higher above and below the last (horizontal) tripwire to alarm. The effects of a previous vertical tripwire alarm are also seen in a right-left bimodality. The target is in one of these four high-probability regions surrounding the intersection of these tripwires. The positive information tracker, however, produces an elliptical distribution centered at the intersection of the tripwires. The averaging effect of the positive information tracker results in a lower Mean Missed Distance, but at the expense of accurately reporting the target likelihood distribution.

Considering only the negative information tracker, the grid pattern produces the best tracking performance. It is important to note, however, that this comparison is made using comparable total *length* of tripwires, without regard to cost. The grid pattern's better performance justifies its choice in favor of the spoke pattern in a budget-limited implementation only if the cost of tripwires is a function of their length.

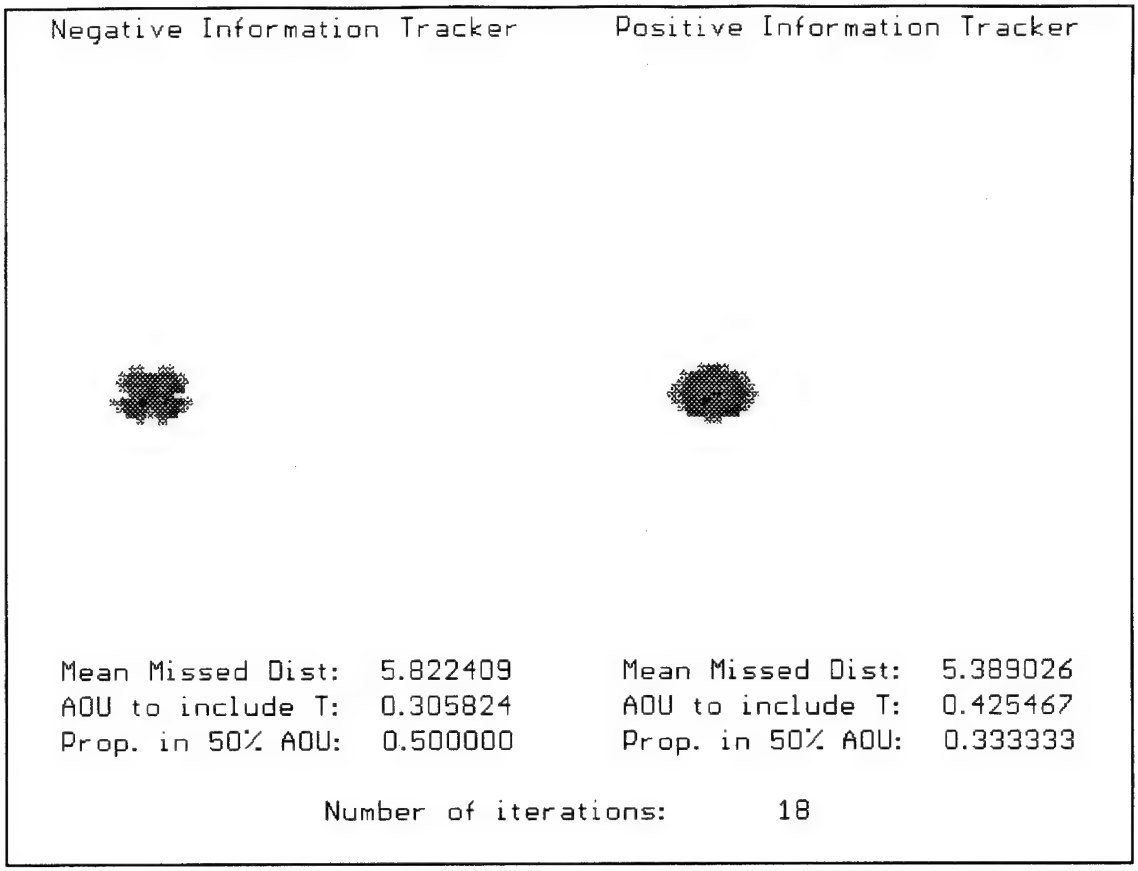

Figure 7. *HexTrack* display demonstrating typical *Grid* pattern tracking. The positive information tracker produces a unimodal distribution with a low MMD; the negative information tracker is more accurate.

B. SENSITIVITY OF TRACKERS TO VELOCITY STATE

The results in the preceding section are not significantly different if the target motion assumptions change to include a velocity state for the target by conditioning movement probabilities on the direction of movement in the previous iteration. Although the trackers continue to ignore target velocity when performing the motion update, the simulated target motion now includes a velocity state. Appendix D describes the incorporation of velocity in the target motion. Table II shows the simulation results.

The negative information tracker accuracy is not as good as when the tracker's assumptions match actual target motion, but accuracy is still above 92% in all cases and always better than the positive information tracker. All other statistics show little change

False Alarm Prob.	Tripwire Arrangement	Average MMD (miles)		Accuracy (%)	
		Negative	Positive	Negative	Positive
0.00	Spoke	10.32*	18.46	93.15*	49.94
	Grid	9.27	9.03*	92.08*	76.98
	Random	17.68*	18.86	94.44*	72.18
0.01	Spoke	18.20*	37.28	93.46*	18.58
	Grid	19.37	18.72*	97.59*	78.75
	Random	27.11*	29.93	93.82*	59.89
0.10	Spoke	48.46*	55.83	94.97*	5.75
	Grid	41.00*	41.07	92.74*	77.92
	Random	47.11*	55.74	95.31*	37.81

* denotes winner

Table II. Tracker performance statistics for the target motion model incorporating a target velocity state. The *negative information tracker* has higher accuracy than the *positive information tracker* in every case. The *positive information tracker* has slightly lower MMD only with the grid pattern with zero or low false alarm probabilities. These results differ little from the cases in which target velocity state was ignored.

from the results in Table I. These results imply that the effects of false alarms and tripwire arrangements are more significant than inclusion of target velocity in the tracking model.

C. SENSITIVITY OF TRACKER TO FALSE ALARM PROBABILITY

In addition to the three levels of false alarm probability shown in the preceding sections, simulation runs for intermediate false alarm levels were conducted for the negative information tracker. In one set of runs, the tracker used the actual probability of false alarms for the information update to evaluate the effect false alarms have on tracking. In the other set of runs the tracker assumed false alarms occurred with probability 0.01 to evaluate the tracker's sensitivity to an inaccurate estimate of false alarm rate. The results are shown in Table III and Figure 8.

False Alarm Probability	Average MMD (miles)		Accuracy (%)	
	Tracker Uses Actual False Alarm Probability	Tracker Uses 0.01 False Alarm Probability	Tracker Uses Actual False Alarm Probability	Tracker Uses 0.01 False Alarm Probability
0.00	11.85	18.20	98.80	92.47
0.01	21.41	21.41	96.91	96.91
0.02	22.19	22.56	97.94	90.71
0.03	25.46	21.63	95.80	85.88
0.04	30.97	29.33	97.62	76.00
0.05	33.84	27.08	98.37	71.28
0.06	33.23	31.07	97.71	56.70
0.07	34.97	41.96	96.76	41.50
0.08	37.76	32.21	95.32	37.62
0.09	42.70	39.76	96.30	32.96
0.10	45.22	51.13	97.77	23.34

Table III. Performance of the negative information tracker at several false alarm probabilities. The MMD statistics depend more on the probability of false alarms than the accuracy of the estimate of this probability. Tracker accuracy, however, is significantly degraded by incorrect assumptions.

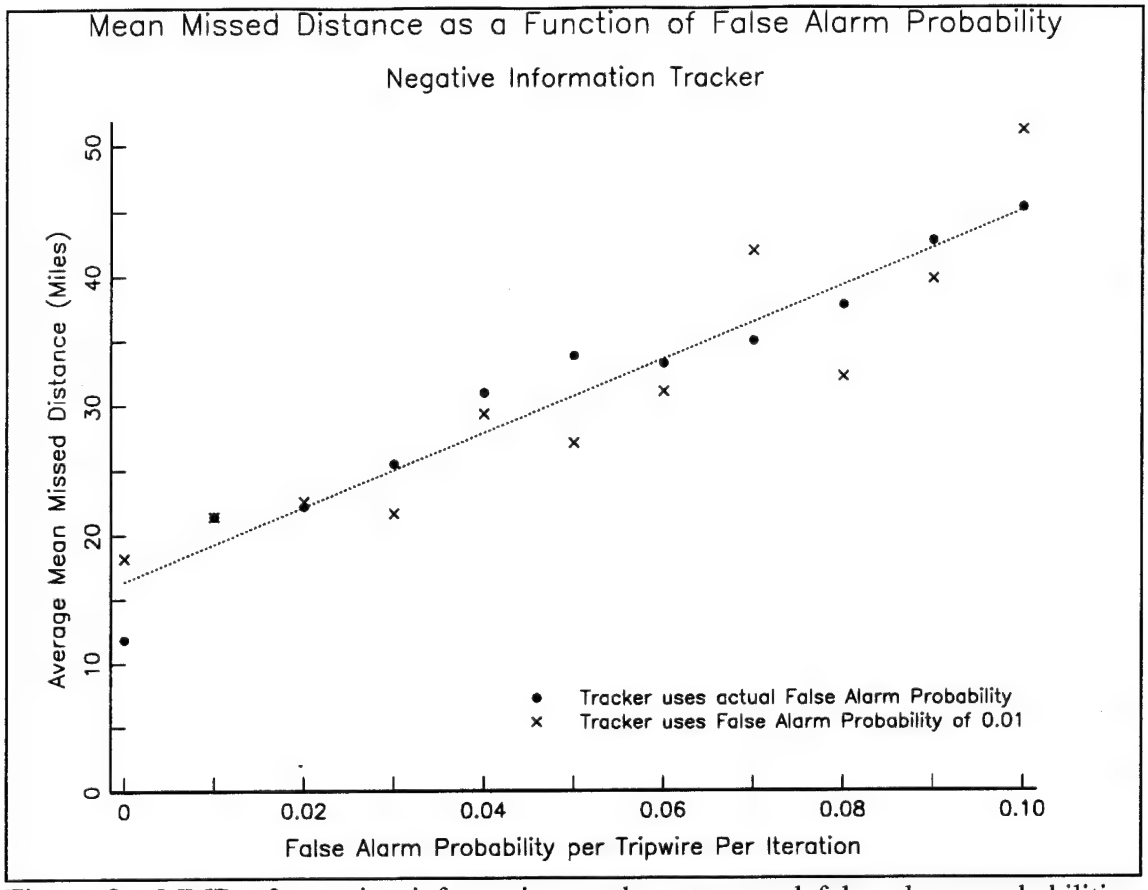

Figure 8. MMD of negative information tracker at several false alarm probabilities. MMD depends more on actual false alarm probability than the accuracy of the estimate.

Surprisingly, there is not a discernable difference in the MMD of the tracker using correct false alarm probabilities, and the MMD using a constant value. This suggests that the actual detections and false alarms are more important in determining MMD than the weight applied to each detection or non-detection, a function of assumed false alarm rate.

The accuracy of the tracker incorrectly assuming a false alarm probability of 0.01, however, shows that the incorrect assumption produces distributions that are increasingly inaccurate as the magnitude of the error increases. Figure 9 shows the effects of the improper assumptions.

When there are no false alarms but the tracker assumes that there are, it does not give enough weight to the alarming tripwires. Although each alarm, actually a detection, should limit the likelihood distribution exclusively to the area covered by the detecting

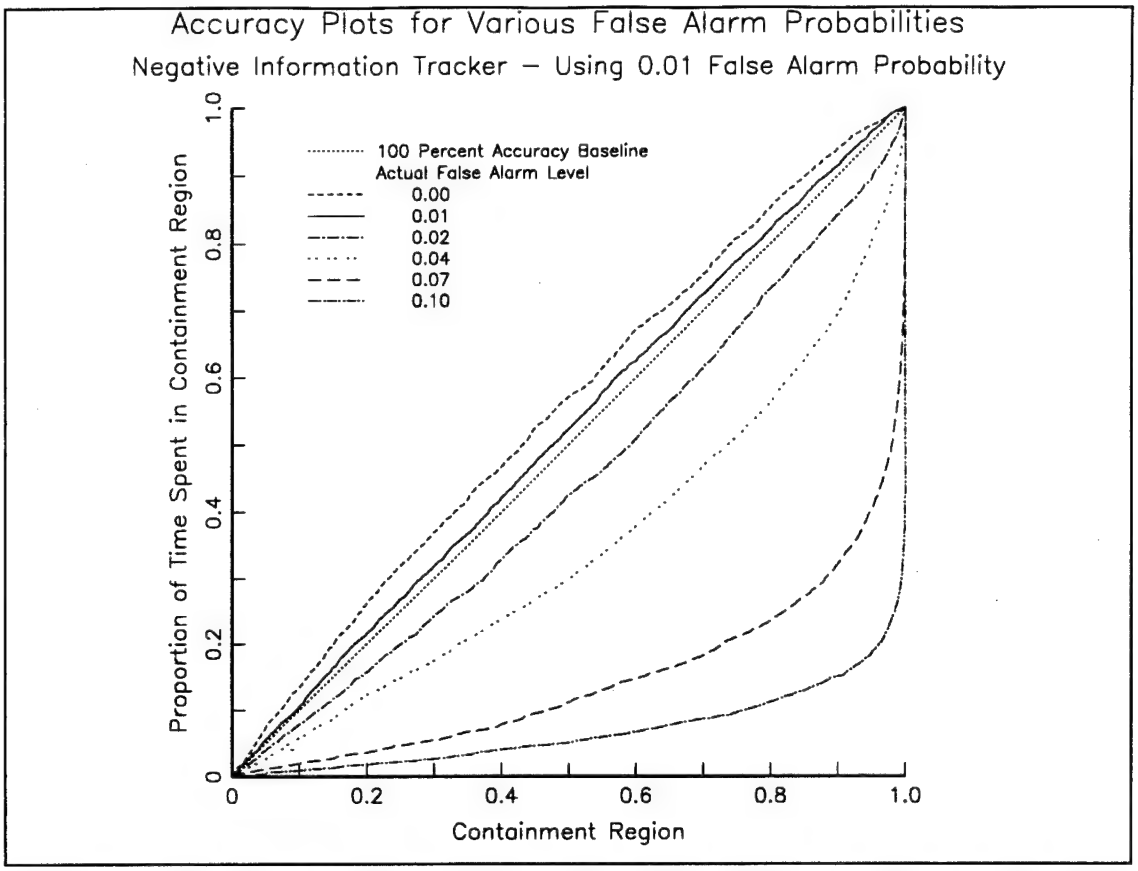

Figure 9. Accuracy of the negative information tracker at several false alarm probabilities. Underestimating false alarm probability produces a pessimistic distribution; overestimating produces an optimistic distribution.

tripwire, the assumed possibility that the detection is false allows some of the distribution to remain away from the detecting tripwire. The probability density of the likelihood distribution in cells near the detecting tripwire is lower than it should be, but is still higher than surrounding areas. A lower probability AOU contains the target, at the detecting tripwire. The result is a pessimistic distribution; the tracker has better localization of the target than it indicates.

When there are more false alarms than assumed, too much weight is given to tripwire alarms, that are also more distributed. The tracker generates a distribution peaked higher at alarming tripwires than is appropriate, requiring larger AOUs to contain a target not near an alarming tripwire. This produces a distribution that is too optimistic.

In an actual implementation of a tripwire field, the rate of false alarms (in the absence of a target) is measurable, so estimation errors are expected to be small. The results of this section show that the actual rate of false alarms are more significant than inaccurate estimation of the false alarm probability.

V. OPTIMAL ARRANGEMENT OF A TRIPWIRE FIELD

The results of Chapter IV demonstrate that *HexTrack*'s negative information tracker produces accurate distributions. This chapter discusses arrangement of a field of tripwires in an optimal cost-effective sense using MMD as a measure of (in-) effectiveness.

There are an infinite number of ways of arranging a field of tripwires. The three arrangements considered in Chapter IV were based on specific features of the many possible arrangements. Although the cost of an array of tripwires is uncertain, tripwire sources cost an estimated 10 to 100 times as much as receivers. This chapter uses a factor of 10 throughout.

The *spoke* pattern, shown in Figure 12 of Appendix A, is based on source-receiver tripwires, taking advantage of the cost savings available when using multiple receivers per source and multiple sources per receiver. The next section introduces a similar pattern based on many receivers arranged around each source, called the *circle* pattern.

The *grid* pattern, shown in Figure 13 of Appendix A, minimizes the size of areas between tripwires, reducing the distance the target travels between successive detections. This pattern is only practical if the cost of each tripwire is primarily a function of length, or if sources can not form tripwires with multiple receivers. Using the source-receiver tripwires, however, the grid pattern becomes prohibitively expensive and wasteful.

The *grid* pattern shown in Figure 13 costs almost seven times as much as the *spoke* pattern. The *spoke* pattern, if repeated over a large area, uses 20 receivers per source. Note that this pattern forms 42 tripwires per source by allowing receivers on the edges of the hexagonal pattern to form tripwires with two sources, and receivers on the corners to form tripwires with three sources. The hexagonal area covered by one source, and its associated tripwires, is 6495 square miles. Using the cost of a receiver as one unit, and assuming sources are ten times more expensive than receivers, each set of 42 tripwires costs 30 units (an average cost of 0.71 units per tripwire). The coverage available per cost unit is 216.5 square miles. To cover the 22,500 square mile area costs

104 cost units. In contrast, the *grid* pattern that has the same cumulative length of tripwires as the *spoke* pattern employs 66 sources and receivers (see Figure 13 in Appendix A) at a cost of 726 units.

Using the *grid* pattern at the same cost as the *spoke* pattern greatly reduces the number of tripwires available and results in large gaps in coverage. If a *grid*-type pattern is desired at the same 104 unit cost (per 22,500 square miles) as the *spoke* pattern, the 104 cost units would purchase an average of 9.5 sources and receivers. If arranged in an alternating pattern along each axis, these sources and receivers would form only 19 tripwires. Since spanning the search area in each direction requires three tripwires, the resulting pattern would have an average of 3.2 grid lines each 150 miles, leaving squares 47 miles on each side. It is not necessary to perform simulation runs to observe that such a pattern would result in poor tracking performance (in terms of MMD).

The *random* arrangement, shown in Figure 14 of Appendix A is used to provide comparison with previous models of tracking performance. Observation of *HexTrack*'s trackers with this arrangement also permits qualitative analysis of tracker performance under various conditions, such as intersecting tripwires or gaps in coverage. The random arrangement, using only one source for each receiver, is the costliest arrangement and does not represent a practical method of distributing tripwires.

A pattern similar to the *spoke* pattern, with many receivers per source, is clearly cost-effective. There are other variables in this type of pattern, however. The following sections discuss variation of the distance between sources, at constant cost, and variation of the number of receivers per source.

A. PROXIMITY OF SOURCES

The *spoke* pattern is based on locating sources exactly close enough to each other such that every point in the search area is within 50 miles (the assumed maximum tripwire length) of a source. The *spoke* pattern is a refined version of a pattern called the *circle* pattern. The *circle* pattern is constructed by arranging receivers around a source

at equal intervals on a circle with radius 50 miles. All source-receiver pairs that are within 50 miles of each other form tripwires.

The figures in Appendix B show the *circle* pattern with sources placed at increasingly large distances from each other. The pattern is more evident viewing Figures 26 through 35 first, followed by Figure 25 and previous figures. The pattern shown in Figure 23 has 20 receivers around each source, with sources arranged at precisely the same distance from each other as in the *spoke* pattern. Note that more than 20 tripwires are associated with each source, as receivers around other sources are within the 50 mile radius. This pattern is essentially a rearrangement of the same number of sources used in the *spoke* pattern.

To evaluate the effect of overlapping coverage by moving sources closer, and of leaving gaps in the coverage by moving sources apart, several additional *circle* patterns were generated. Using the proximity of sources to each other in the *spoke* pattern as a reference point of 1.0, the number of receivers per source for other proximities is calculated assuming constant total cost, and a repeating pattern over a large area. As the sources become closer together, more are necessary to cover an equivalent search area, so fewer receivers can be purchased. Having receivers from other sources close enough to form tripwires partially mitigates this reduction in receivers. As sources are moved apart, the savings from fewer sources allow the purchase of more receivers per source. However, each receiver is only within range of one source. Appendix B shows the tripwires formed at each level of proximity. Note that all of the *circle* patterns cost the same as the *spoke* pattern.

Figure 10 and Table IV show the effect of varying source proximity. Relative Proximity represents the factor of overlapping (Relative Proximity less than 1.0) or gapping (Relative Proximity greater than 1.0) coverage. The pattern with proximity 1.0 has 20 receivers per source and has sources arranged exactly far enough that every point in the search area is within the maximum tripwire length. False alarms occur at a probability of 0.01 per tripwire per iteration.

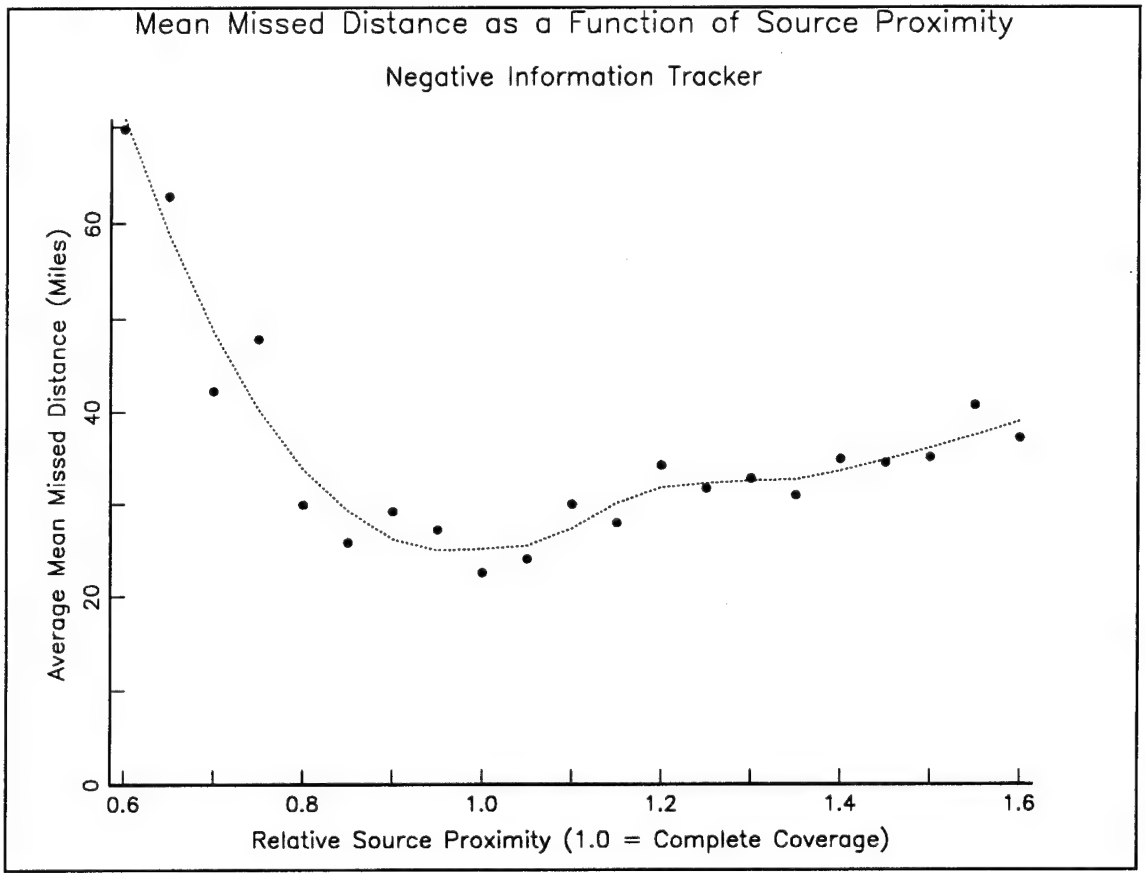

Figure 10. MMD of the negative information tracker as a function of source proximity in the *circle* pattern. Moving sources together allows overlapping coverage, but results in fewer receivers and some shorter tripwires. Moving sources apart permits more receivers, but causes gaps in coverage.

Relative Proximity	Receivers per Source	Average MMD (miles)
0.60	1	69.78
0.65	3	62.83
0.70	5	42.26
0.75	7	47.87
0.80	9	29.91
0.85	12	25.78
0.90	14	29.17
0.95	17	27.16
1.00	20	22.54
1.05	23	24.02
1.10	26	29.93
1.15	30	27.94
1.20	33	34.15
1.25	37	31.66
1.30	41	32.74
1.35	45	30.93
1.40	49	34.84
1.45	53	34.44
1.50	57	35.05
1.55	62	40.74
1.60	67	37.15

Table IV. MMD of the negative information tracker as a function of source proximity in the *circle* pattern. The lowest MMD is at a Relative Proximity of 1.0, where sources are at the same proximity as the *spoke* pattern.

At the low extreme, the cost of the additional sources drastically reduces the number of receivers per source, resulting in few tripwires and large areas of uncertainty. As the receiver-per-source ratio increases, MMD rapidly decreases as more tripwires form. As gaps in coverage appear and receivers begin to be in range of only one source, MMD increases sharply until all receivers are associated with only one source. Increasing gaps in coverage cause the remaining increase in MMD, although the increased number of receivers per source mitigates the increase. Figure 10 indicates that proximities near 1.0 are the lowest. The optimal *circle* pattern MMD of 22.54 miles is also comparable with the MMD of the same-cost *spoke* pattern under otherwise identical assumptions, 21.41 miles.

B. DENSITY OF TRIPWIRE FIELD

Concluding from the previous section that the spoke pattern is a near-optimal arrangement, this section considers the marginal benefit of additional receivers. The number of tripwires per source in the spoke arrangement was varied from six to 60. Figure 11 and Table V show the results of this analysis.

These patterns are not constant cost; they use the same number of sources in the same location, while varying the receiver-per-source ratio to produce more tripwires. For example, the 12 tripwire-per-source pattern costs half as much as the 42 tripwire-per-source *spoke* pattern used in Chapter IV. A ten to one ratio of source to receiver cost is assumed.

Significant improvement in tracker performance is evident up to about 30 tripwires per source, with smaller marginal improvement from addition of further tripwires. Evaluation of the tactical requirements and value of added effectiveness is necessary to state an optimal cost-effective value, but the results suggest a minimum of 30 tripwires per source to take advantage of the significant improvement in performance.

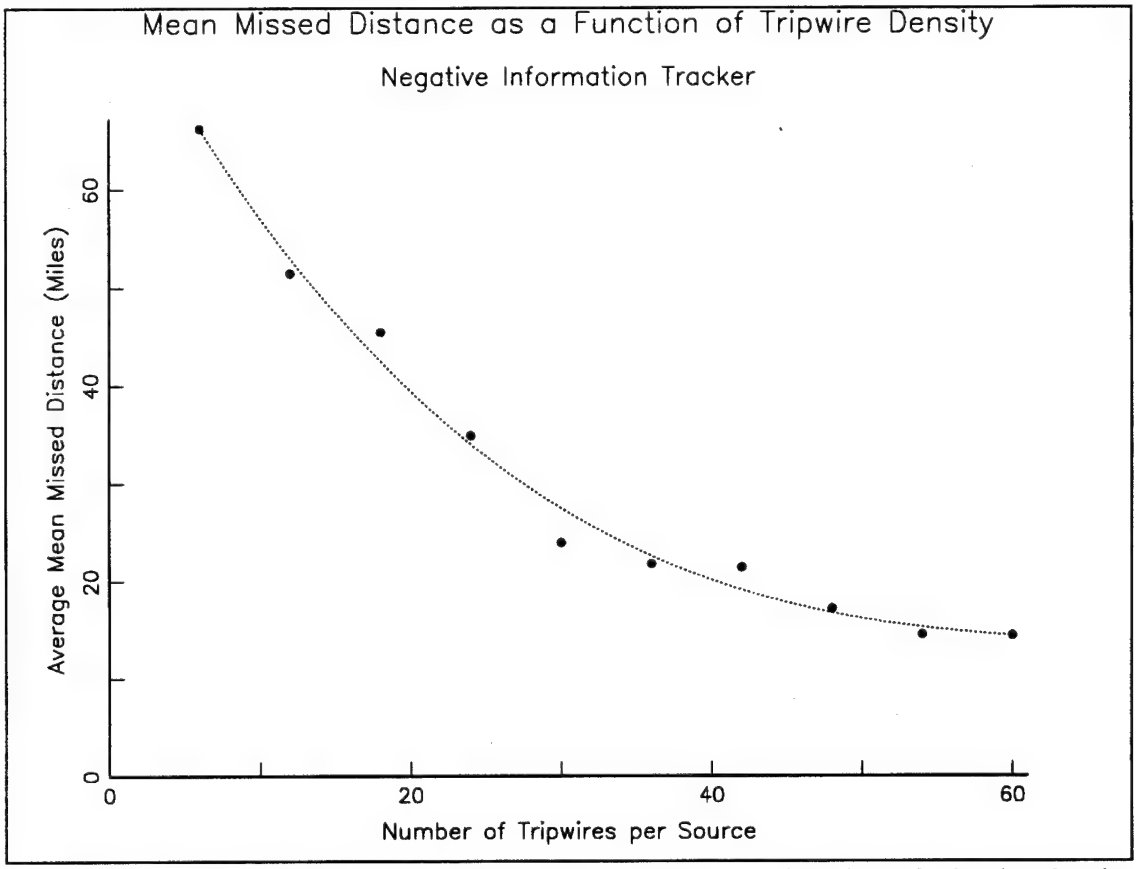

Figure 11. MMD of the negative information tracker as a function of tripwire density in the *spoke* pattern. Significant improvement occurs up to 30 tripwires per source; marginal improvement occurs at higher numbers.

Tripwires per Source	Receivers per Source	Cost Relative to Spoke Pattern	Mean Missed Distance (miles)
6	2	0.4	66.10
12	5	0.5	51.52
18	8	0.6	45.41
24	11	0.7	34.87
30	14	0.8	23.92
36	17	0.9	21.76
42	20	1.0	21.41
48	23	1.1	17.14
54	26	1.2	14.49
60	29	1.3	14.37

Table V. MMD of the negative information tracker as a function of tripwire density in the *spoke* pattern. Significant improvement occurs up to 30 tripwires per source; marginal improvement occurs at higher numbers.

VI. CONCLUSIONS

A. SIGNIFICANCE OF NEGATIVE INFORMATION

Fusion of negative information into a tracking algorithm can significantly enhance its performance. The effect is most distinct in arrangements of line-of-position sensors that share a common point, and in situations where false alarms produce a large number of detections. In the case of forward scatter arrays, where cost-effective arrangements of tripwires have coincident endpoints, fusion of negative information into the target motion analysis is necessary to produce accurate results.

Implementation of tracking using negative information requires estimates of the probabilities of target detection as well as false alarms. While improper assumptions degrade tracker performance, the effect of these assumptions is less significant than choosing not to take negative information into account.

B. TRIPWIRE ARRANGEMENT

In a source-receiver implementation of tripwires, practical implementation of an array of tripwires would involve placement of many receivers around each source. Constant-cost analysis of the effects of leaving gaps in, or overlapping, coverage, shows that arrangement of the sources just far enough apart to provide complete coverage results in the best localization of the target. The spoke arrangement implements this coverage plan while also using multiple sources per receiver.

Addition of tripwires can continually enhance the performance of a field of tripwires. There appears to be a minimum number of tripwires needed to provide adequate tracking capability, with further additions marginally improving performance.

C. SUGGESTIONS FOR FURTHER RESEARCH

This thesis makes several simplifying assumptions that warrant further analysis to extend the results to a wider range of real-world problems. One significant assumption is that of a single target in the search area. Further analysis could develop MOEs and

analyze the significance of negative information in a no-target or multiple target environment.

The trackers in this thesis did not include a velocity state for the target because of programming constraints in the microcomputer implementation. Although analysis shows this effect to be small, further study into how a negative information tracking analysis can include target velocity state could provide additional useful results.

Finally, this thesis analyzed a specific sensor, the forward scatter tripwire. The significance of negative information for other types of sensors is not obvious. A model fusing both positive and negative information from multiple sensors of varying types can extend the results of this thesis to many tracking scenarios.

APPENDIX A. CONSTANT CUMULATIVE LENGTH TRIPWIRE PATTERNS

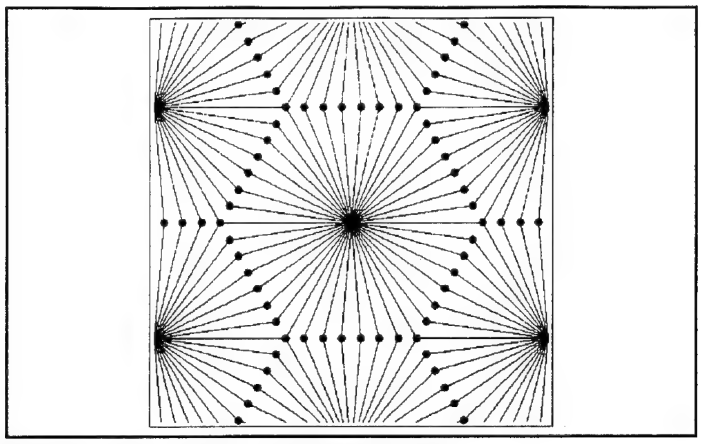

Figure 12. *Spoke* tripwire pattern.

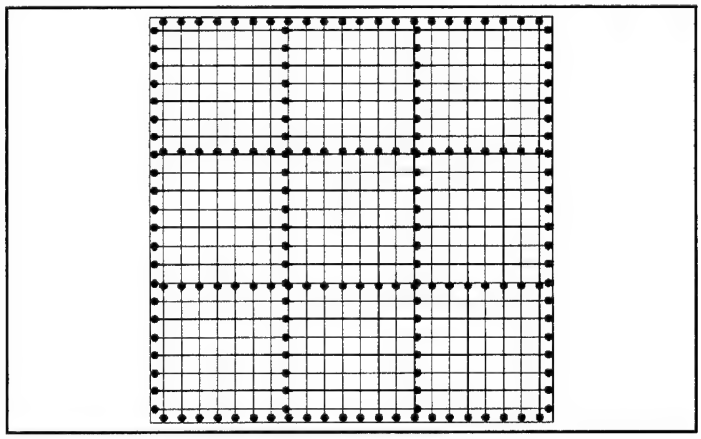

Figure 13. *Grid* tripwire pattern.

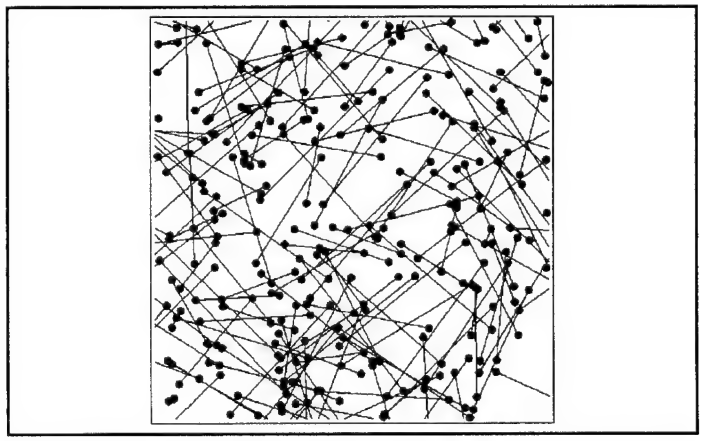

Figure 14. *Random* tripwire pattern.

APPENDIX B. VARIABLE SOURCE PROXIMITY TRIPWIRE PATTERNS

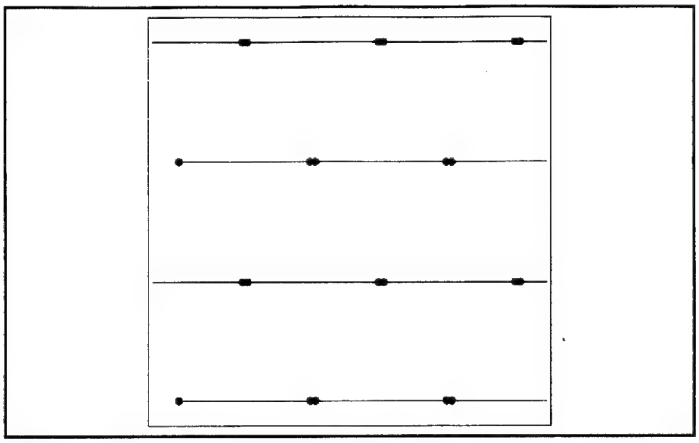

Figure 15. 0.60 Relative Proximity *Circle* pattern.

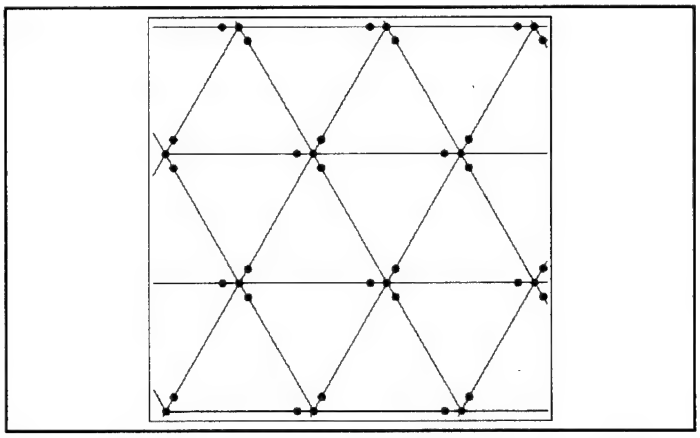

Figure 16. 0.65 Relative Proximity *Circle* pattern.

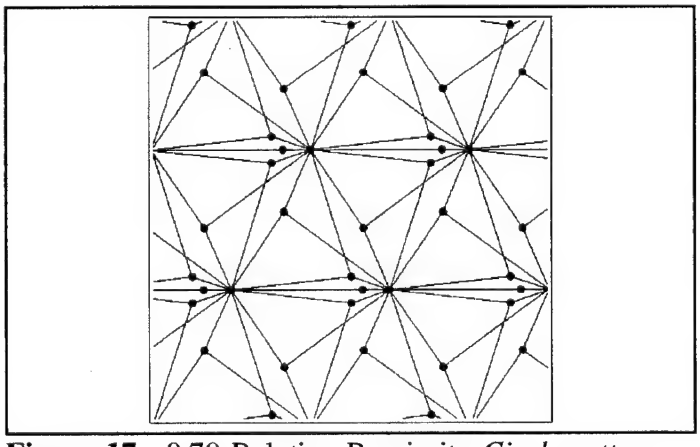

Figure 17. 0.70 Relative Proximity *Circle* pattern.

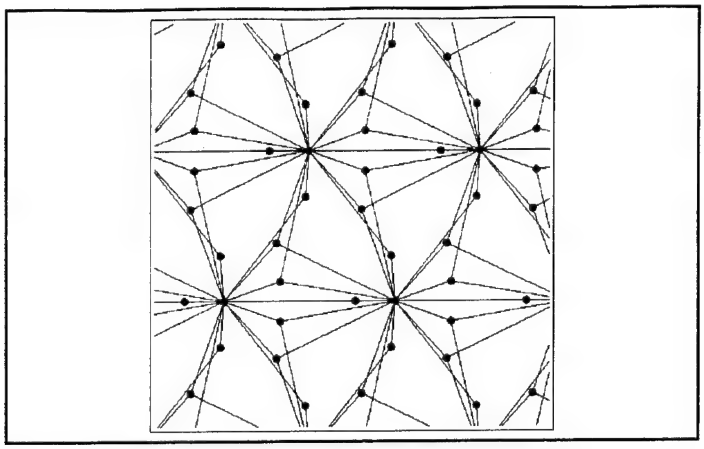

Figure 18. 0.75 Relative Proximity *Circle* pattern.

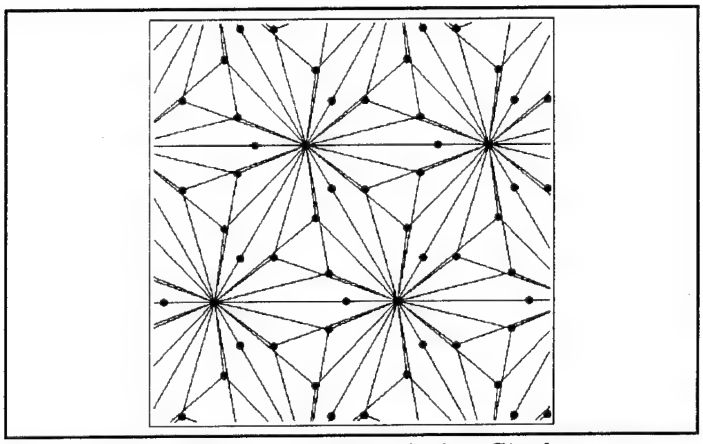

Figure 19. 0.80 Relative Proximity *Circle* pattern.

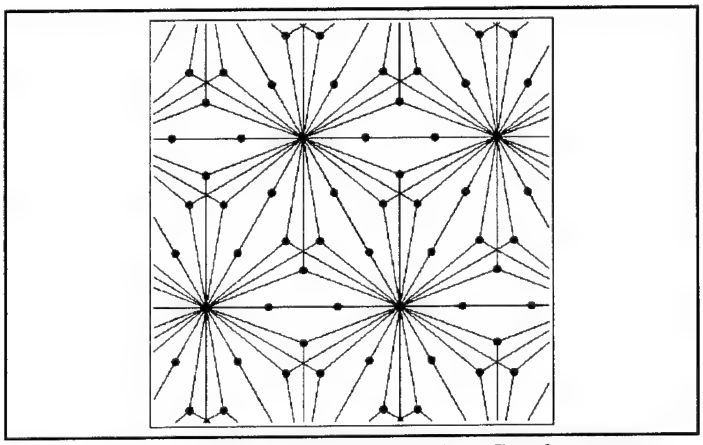

Figure 20. 0.85 Relative Proximity *Circle* pattern.

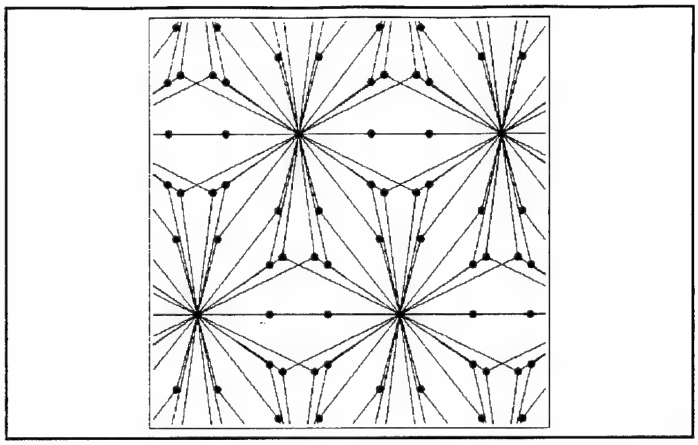

Figure 21. 0.90 Relative Proximity *Circle* pattern.

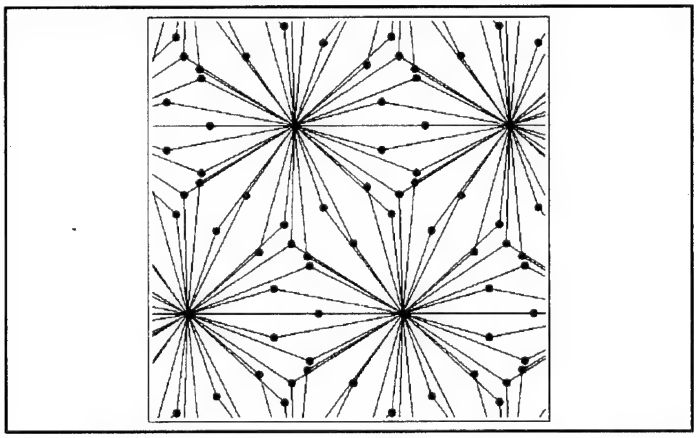

Figure 22. 0.95 Relative Proximity *Circle* pattern.

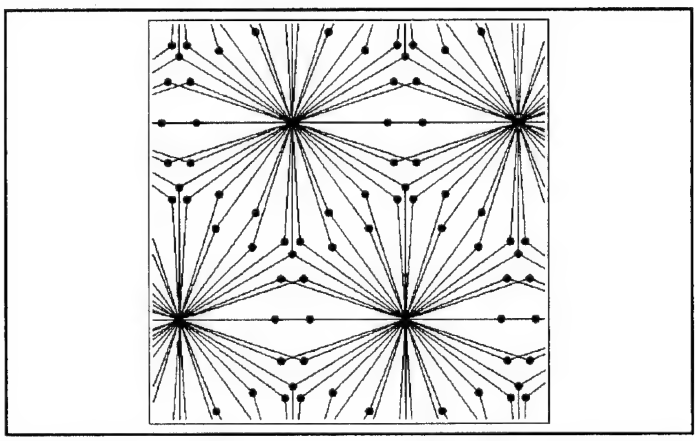

Figure 23. 1.00 Relative Proximity *Circle* pattern.

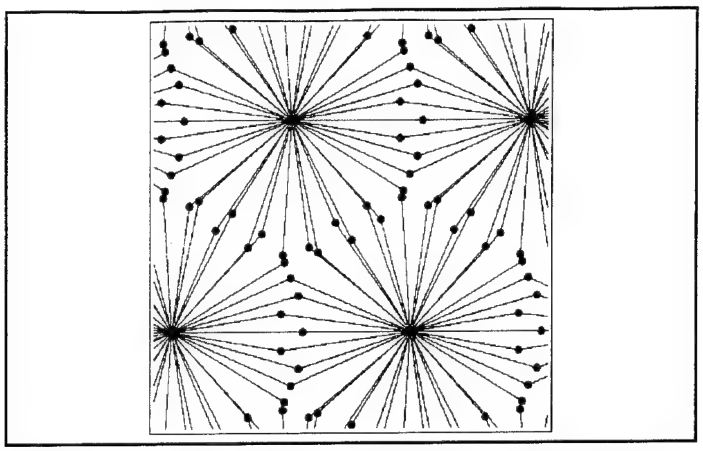

Figure 24. 1.05 Relative Proximity *Circle* pattern.

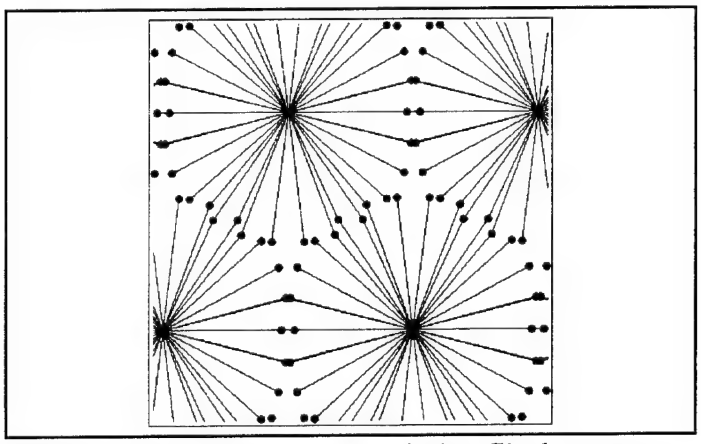

Figure 25. 1.10 Relative Proximity *Circle* pattern.

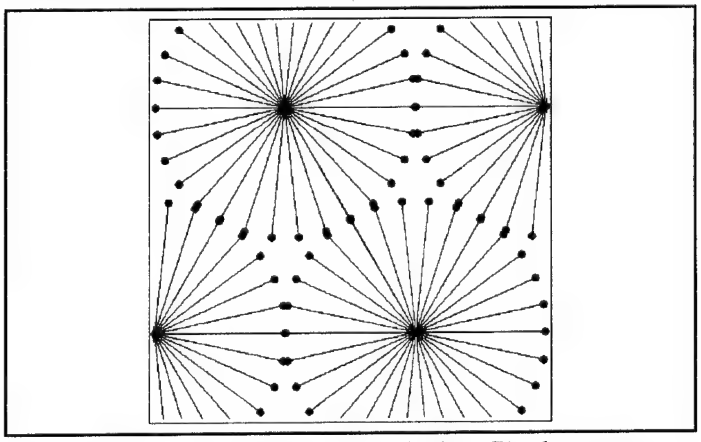

Figure 26. 1.15 Relative Proximity *Circle* pattern.

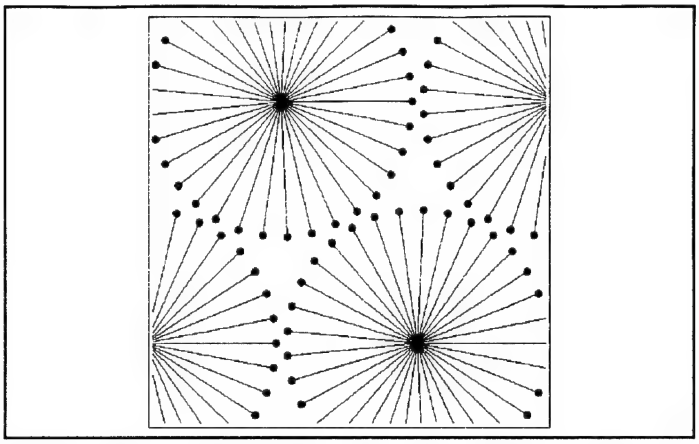

Figure 27. 1.20 Relative Proximity *Circle* pattern.

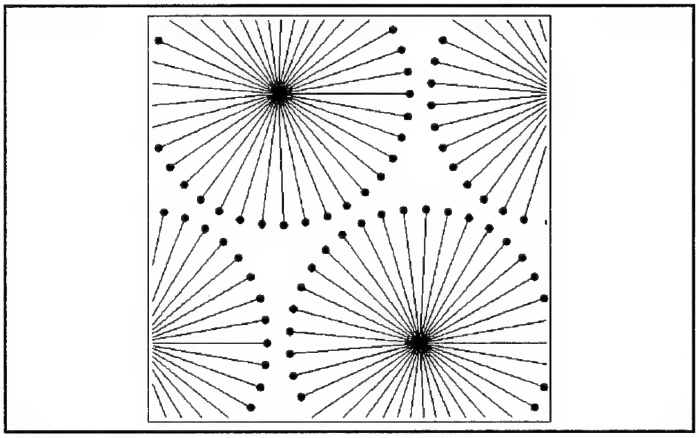

Figure 28. 1.25 Relative Proximity *Circle* pattern.

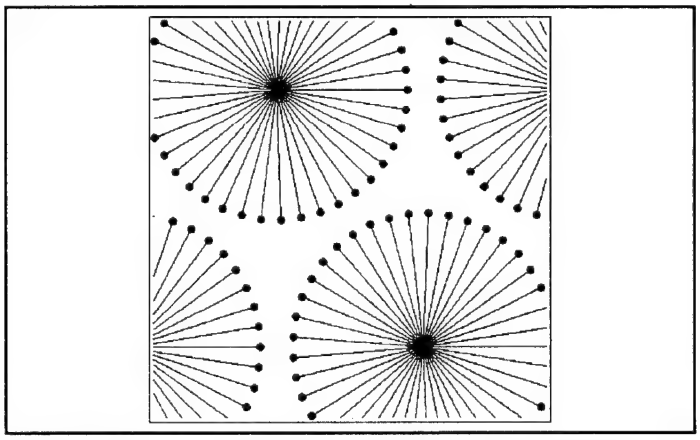

Figure 29. 1.30 Relative Proximity *Circle* pattern.

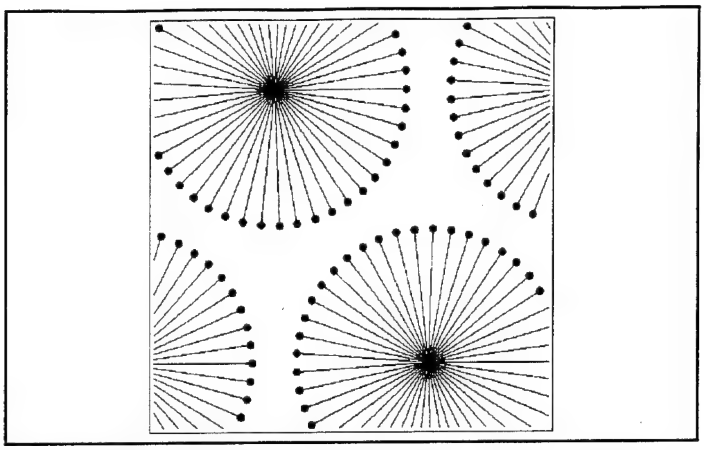

Figure 30. 1.35 Relative Proximity *Circle* pattern.

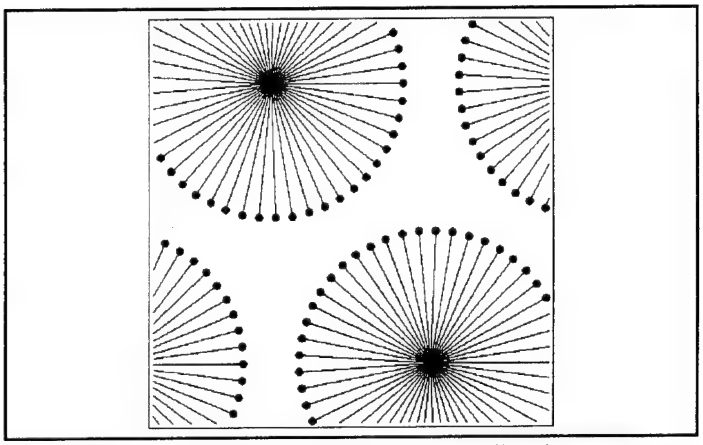

Figure 31. 1.40 Relative Proximity *Circle* pattern.

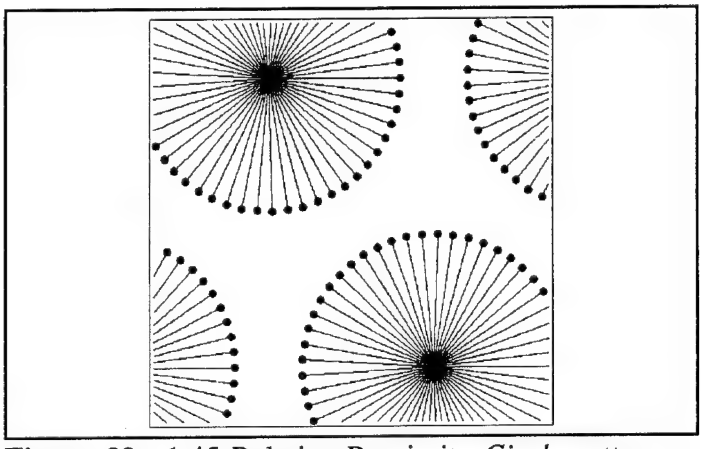

Figure 32. 1.45 Relative Proximity *Circle* pattern.

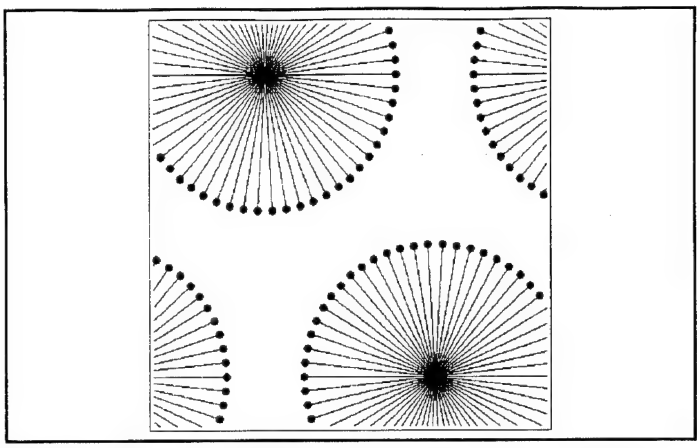

Figure 33. 1.50 Relative Proximity *Circle* pattern.

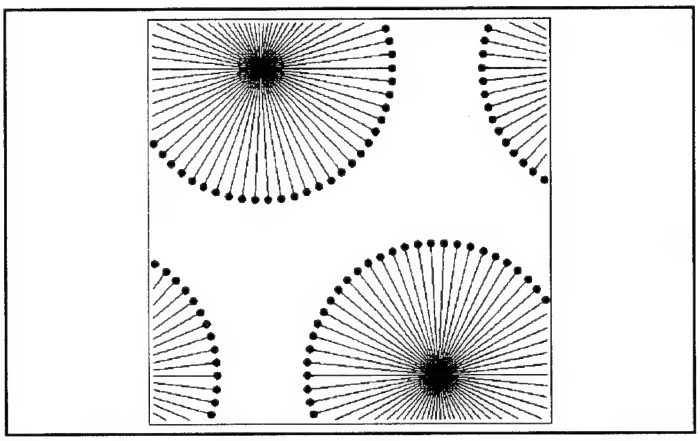

Figure 34. 1.55 Relative Proximity *Circle* pattern.

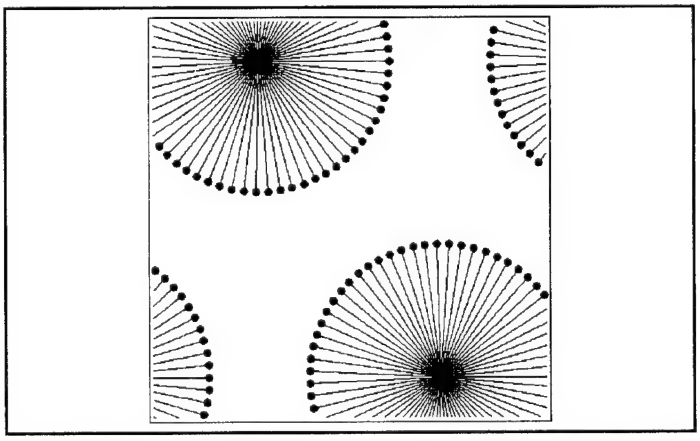

Figure 35. 1.60 Relative Proximity *Circle* pattern.

APPENDIX C. VARIABLE DENSITY SPOKE TRIPWIRE PATTERNS

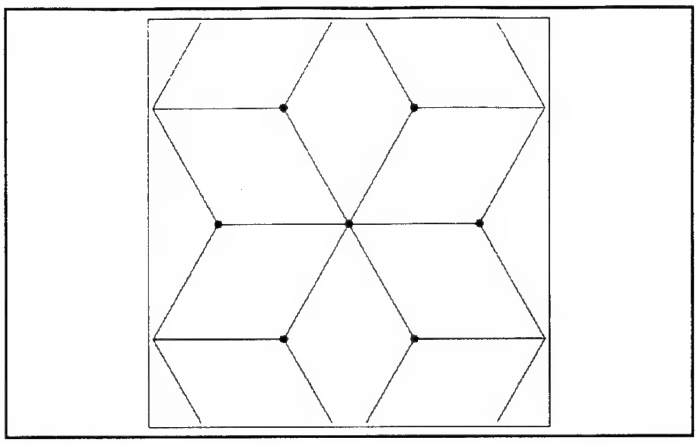

Figure 36. Six Tripwire per Source *Spoke* pattern.

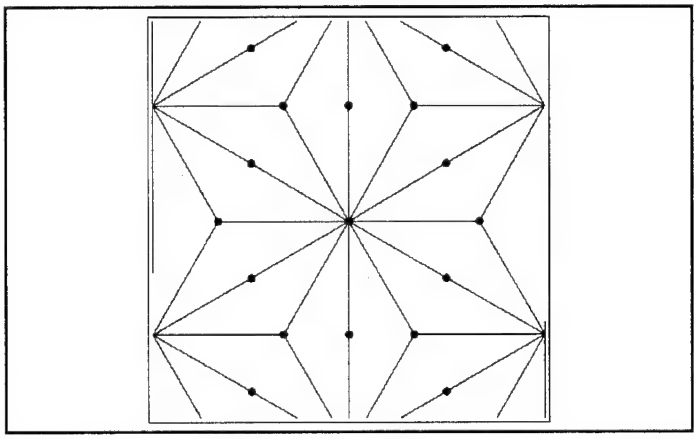

Figure 37. 12 Tripwire per Source *Spoke* pattern.

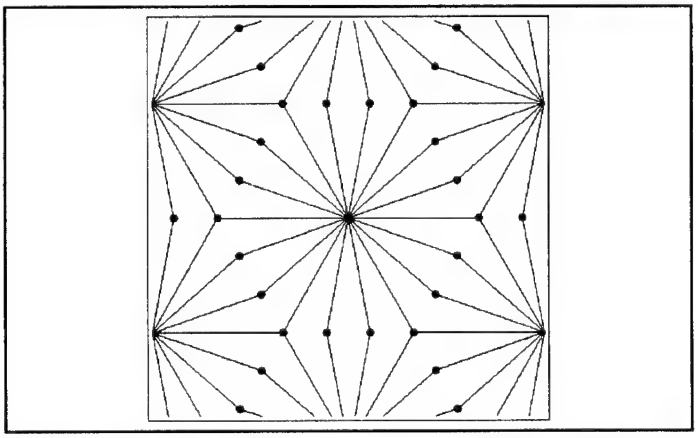

Figure 38. 18 Tripwire per Source *Spoke* pattern.

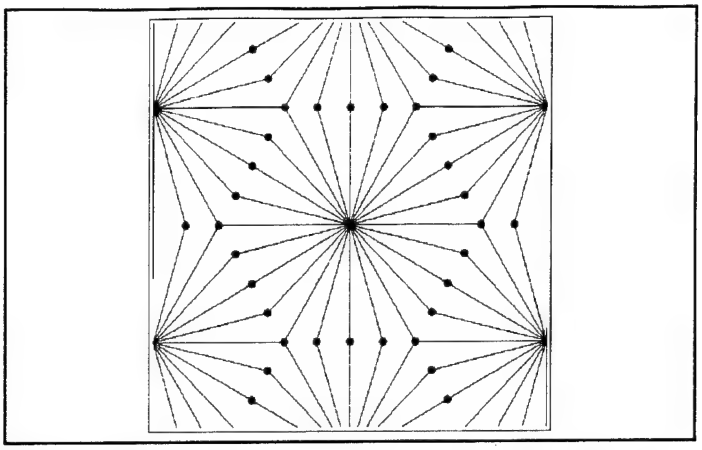

Figure 39. 24 Tripwire per Source *Spoke* pattern.

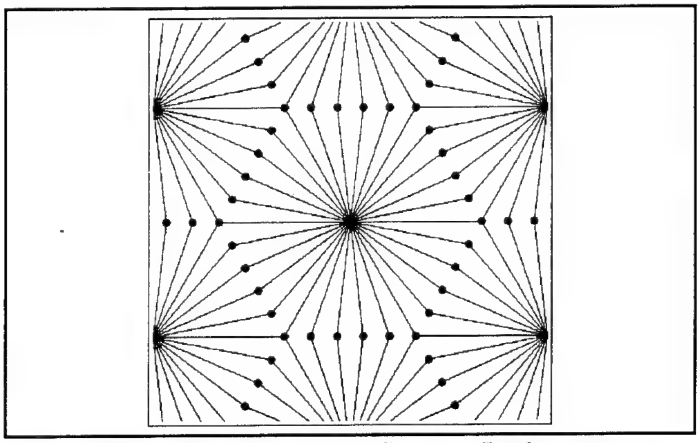

Figure 40. 30 Tripwire per Source *Spoke* pattern.

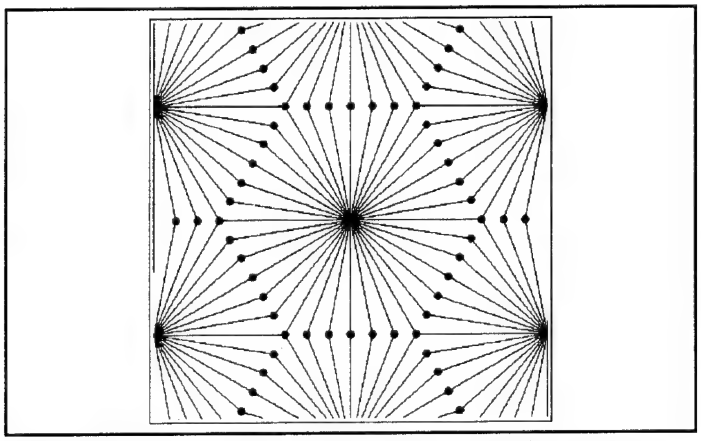

Figure 41. 36 Tripwire per Source *Spoke* pattern.

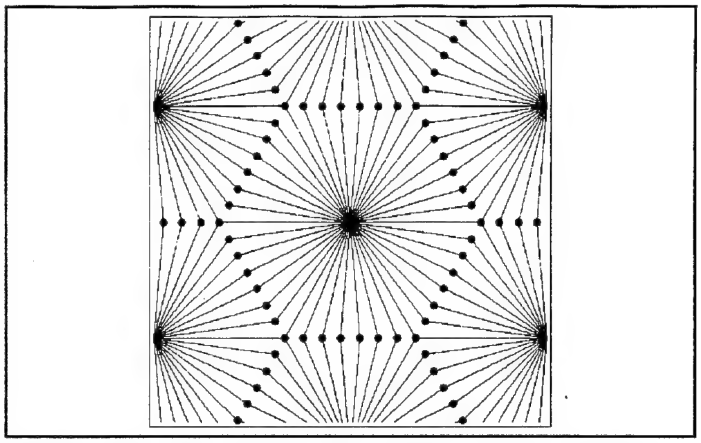

Figure 42. 42 Tripwire per Source *Spoke* pattern.

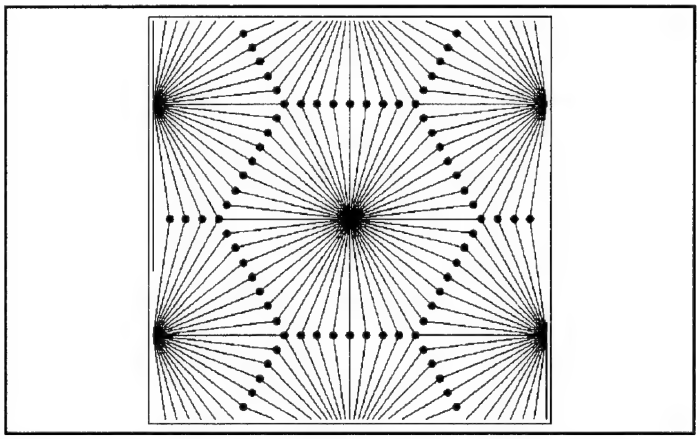

Figure 43. 48 Tripwire per Source *Spoke* pattern.

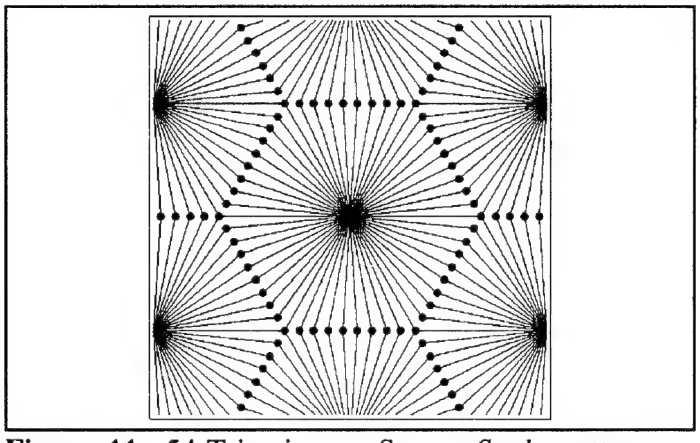

Figure 44. 54 Tripwire per Source *Spoke* pattern.

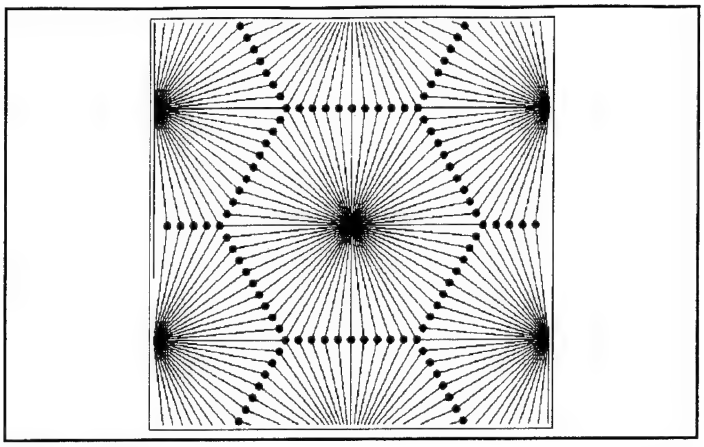

Figure 45. 60 Tripwire per Source *Spoke* pattern.

APPENDIX D. *HEXTRACK* TARGET MOTION ASSUMPTIONS

The target motion model used by *HexTrack* is a discretization of a stochastic target motion model, programmed in another program called *HexMotion.* The target is assumed to execute maneuvers randomly according to a Poisson process, with a rate of three maneuvers per hour. When a maneuver occurs, the target chooses a new course (uniformly from 0° to 360°) and a new speed V where V=5 with probability 0.4, V=10 with probability 0.3, and V=15 with probability 0.3.

HexMotion initializes the target in the center of a cell, and calculates the target's position every 7.06 minutes, corresponding to the discrete tracking intervals in *HexTrack.* Each interval, *HexMotion* determines whether the target remains in the same cell, or enters one of the adjacent cells. This process repeats for 15 million iterations to produce an empirical probability distribution for target motion. Both unconditional probabilities and probabilities conditioned on a change of cell in the previous iteration are calculated.

HexMotion implements the following pseudo-code algorithm:

```
initialize target position and number of iterations;
generate random course and speed;
generate time-to-maneuver (exponential random variable);
time-to-iteration ← 7.06 minutes;

while iterations ≤ 15 million do
        if time-to-maneuver ≤ time-to-iteration then
                calculate new target position at maneuver time;
                generate new random course and speed;
                decrement time-to-iteration by time-to-maneuver;
                generate new time-to-maneuver (exponential random variable);
        else
                calculate new target position at end of iteration;
                determine and record change of cell since last iteration;
                decrement time-to-maneuver by time-to-iteration;
                time-to-iteration ← 7.06 minutes;
                increment iterations;
        end if;
end while;
```

Figure 46 shows the numbering of the cells used in Table VI. Table VI shows the probabilities of movement to adjacent cells conditioned on movement in the previous iteration. *HexTrack* uses these empirical probabilities in its target motion model along with the velocity state. Table VI also shows the unconditional probabilities of movement to adjacent cells. *HexTrack* uses these in its basic motion model and the tracker motion update. The *HexMotion* results for symmetric entries are averaged.

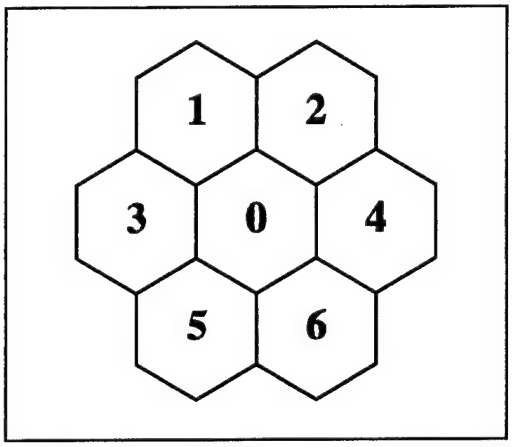

Figure 46. Adjacent cell numbering used in target motion matrix. Cell *n* is symmetric with cell 7-*n*.

Conditional Probabilities		Target Cell Change This Iteration: Cell 0 to ...						
		0	**1**	**2**	**3**	**4**	**5**	**6**
Target Cell Change Previous Iteration: Cell 0 to ...	**0**	.3922	.1013	.1013	.1013	.1013	.1013	.1013
	1	.4316	.1485	.1419	.1419	.0503	.0503	.0355
	2	.4316	.1419	.1485	.0503	.1419	.0355	.0503
	3	.4316	.1419	.0503	.1485	.0355	.1419	.0503
	4	.4316	.0503	.1419	.0355	.1485	.0503	.1419
	5	.4316	.0503	.0355	.1419	.0503	.1485	.1419
	6	.4316	.0355	.0503	.0503	.1419	.1419	.1485
Unconditional Probabilities		**Target Cell Change Every Iteration: Cell 0 to ...**						
		0	**1**	**2**	**3**	**4**	**5**	**6**
		.4150	.0975	.0975	.0975	.0975	.0975	.0975

Table VI. Target motion probabilities used in the *HexTrack* motion model. Conditional probabilities incorporate the target's velocity, represented by change of cell, on the previous iteration. For example, if the target traveled left one cell (from 0 to 3) on the previous iteration, it will move to the next cell to the left (from 0 to 3) this iteration with probability 0.1485. If velocity state is ignored, the target travels one cell to the left with probability 0.0975.

APPENDIX E. *HEXTRACK* SOURCE CODE

```
program HexTrack;
{Author: Daniel B. Widdis}

{$N+ enable 80x87 mode for double-precision reals}

uses Graph,CRT;

type
   MotionArray = Array[0..7, 0..6] of single;

const
   BGIdir = '\tp\bgi';                          {location of BGI files}
   TWFNameS = '\tp\tripwire\spoktrip.dat';      {tripwire data files}
   TWFNameR = '\tp\tripwire\randtrip.dat';
   TWFNameG = '\tp\tripwire\gridtrip.dat';
   TWFNameP = '\tp\tripwire\proxtrip.dat';

   {this is a state transition matrix for target velocity.  Velocity states
    are:   1   2
         3  0  4
          5   6   .  Unconditionals are overall proportion of time in states.}
   TMotion: MotionArray =
     {Conditional: Velocity State}
     { 0         1         2         3         4         5         6  }
{0}((0.3922 , 0.1013 , 0.1013 , 0.1013 , 0.1013 , 0.1013 , 0.1013),
{1} (0.4316 , 0.1485 , 0.1419 , 0.1419 , 0.0503 , 0.0503 , 0.0355),
{2} (0.4316 , 0.1419 , 0.1485 , 0.0503 , 0.1419 , 0.0355 , 0.0503),
{3} (0.4316 , 0.1419 , 0.0503 , 0.1485 , 0.0355 , 0.1419 , 0.0503),
{4} (0.4316 , 0.0503 , 0.1419 , 0.0355 , 0.1485 , 0.0503 , 0.1419),
{5} (0.4316 , 0.0503 , 0.0355 , 0.1419 , 0.0503 , 0.1485 , 0.1419),
{6} (0.4316 , 0.0355 , 0.0503 , 0.0503 , 0.1419 , 0.1419 , 0.1485),
    {Unconditional}
    (0.4150 , 0.0975 , 0.0975 , 0.0975 , 0.0975 , 0.0975 , 0.0975));

   {this constant determines detection probability of a single tripwire
    in a single iteration for a target within its detection region}
   DetectProb = 0.8;

   {these constants are used for statistical data collection}
   Iterations = 50;
   Replications = 60;
   StopTime = Iterations*Replications;

   {these constants define size of hex grid}
   GridSize = 150;                      {miles, each dimension}
   YCells = 85;                         {max:85 due to heap overflows}
   XCells = Round(YCells*0.866025404); {85*cos(PI/6) = 74}
   Cells = XCells*YCells;               {85*74 = 6290}

   Radius = (GridSize*2/3)/(YCells+0.5);  {miles, from center of hex
                                           to each corner}
   Scale  = 312/((XCells+0.5)*1.732050808);  {pixels per hex}

   MaxTW = 250;     {max number of tripwires (actual number from TWFile)}

type
   {This type is for TWArray, used to calculate detection probabilities
    for each cell-tripwire combination}
   TWPtr = ^TWRecord;
   TWRecord = record
                X1, Y1, X2, Y2 : single; {tripwire endpoints}
              end; {record}
   TWArrayType = Array[1..MaxTW] of TWPtr;
```

```
    {This type is for TWlist, a part of each CellRecord, a linked list of
     tripwires 'incident' (close enough for nonzero detection probability)
     to each cell}
    IncTWPtr = ^IncTWRecord;
    IncTWRecord = record
                    TWNum : Integer;   {incident tripwire}
                    AlmProb : double;  {probability of detection
                                        OR false alarm}
                    Next : IncTWPtr;   {next incident one}
                  end; {record}

    {This type is for CellArray}
    CellPtr = ^CellRecord;
    CellRecord = record
                   CellX,
                   CellY : single;     {cell center, used for detection
                                        probability calcs and missed dist.}
                   Ndensity,           {target distribution density with
                                        negative information considered}
                   Pdensity,           {target distribution density without
                                        negative information considered}
                   Temp,               {for Nd, Pd calculations}
                   NAP    : double;    {probability no tripwires alarm
                                        given target in this cell}
                   TWlist : IncTWPtr; {list of TWs incident to cell}

                   {Note: Sort vars index other cells:
                       CellArray[1]^.NSort indexes cell with highest
                       Ndensity (similar for PSort)}
                   NSort,              {for AOU calcs of Ndensity}
                   PSort  : Integer;  {for AOU calcs of Pdensity}
                 end; {record}

    CellArrayType = Array[1..Cells] of CellPtr;
    AdjCellType = Array[1..6] of Integer; {used for movement update}
    HexType = Array[1..12] of Integer;    {used to draw hexes on screen}

var
    CellArray : CellArrayType;
    TWArray : TWArrayType;
    AdjCells : AdjCellType;          {list of adjacent cell numbers (temp)}

    TargetC,                         {'real' target cell}
    Check,                           {file input line check}
    NumTW,                           {number of TW's read in}
    X, Y, C,
    NextMatch,
    TC, HeapSize : Integer;          {counting variables}

    LastMove,                        {velocity state}
    Color : Byte;                    {used for graphical display of likelihood
                                      distributions}
    Dist,
    TWLen, Proj,
    Prob, RadEq : single;            {for tripwire detection calcs}
    FalseProb,                       {false alarm probability of a single
                                      tripwire in a single iteration}
    FalseProbAssumed,                {false alarm probability used in tracker}
    NTotalProb,
    PTotalProb : double;             {for normalizing/AOU size calcs}

    NMMD,
    PMMD,                            {Mean missed distance}
    NContain,
    PContain : single;               {Prob in this or higher prob cell}
    NIn50AOU,
    PIn50AOU,                        {Iterations target contained in 50%
```

```
                                    containment region}
   SimTime : Integer;               {Total elapsed iterations}

   TempPtr,                         {for incident TW incrementing}
   TempPtr2 : IncTWPtr;             {for other cells incident to TW}
   Alarmed,                         {boolean if TW alarms}
   Searching : Boolean;             {when TW found in list}

   SimFile,                         {stats output}
   TWFile : Text;                   {stores tripwire pattern}

   SimFName,
   TWFName,                         {selected tripwire field}
   S          : STRING;             {for graphic text output}
   PolyPoints : HexType;            {for FillPoly procedure}
   GraphicsOn : BOOLEAN;            {toggle display of densities}
   InitStatus : Integer;            {temp variable to watch progress}
   CH : CHAR;

{-------------------------------------------------------------------------}
{ GRAPHICS INITIALIZATION                                                 }
{-------------------------------------------------------------------------}
procedure InitGraphics;
   {Initializes graphics mode; returns error if not successful.}
   {This procedure from D. Cooper, _Oh! Pascal!_ 1992, p. 111.}

   var GraphDriver,
       GraphMode:  Integer;

   begin {InitGraphics}
      GraphDriver := Detect;
      InitGraph (GraphDriver, GraphMode, BGIdir);
   end; {InitGraphics}
{-------------------------------------------------------------------------}
{ BINARY HEAP SORTING ROUTINES                                            }
{-------------------------------------------------------------------------}
{ The Heapify and HeapExtractMax routines are used in a binary heapsort
  routine.  In the heapsort, CellArray begins with NSort and PSort
  referencing cells sorted by the values of NDensity or PDensity on the
  previous iteration.  To update this sort, the binary parents are heapified
  from the bottom to the top of the heap, placing lower NDensity and PDensity
  at the top.  The HeapExtractMax routine is then called to sequentially
  remove the lowest value and place it at the end of the list, reducing the
  size of the remaining heap.  The resulting array is sorted such that
  CellArray[CellArray[1]^.NSort]^.NDensity is the highest density.}

procedure NHeapify(var CellArray: CellArrayType; i : integer);
{Maintains binary heapsort property for i and its children.
 Assumes 'children' of i have heap property.}

   var
      L,
      R,
      M,
      Temp : integer;

   begin {NHeapify}
      L := i SHL 1;             {2i}
      R := (i SHL 1) OR 1;      {2i+1}
      if L <= Heapsize then begin {if I is leaf, exit}
         M := L;
         if R <= Heapsize then begin
            if CellArray[CellArray[L]^.NSort]^.Ndensity >
                  CellArray[CellArray[R]^.NSort]^.Ndensity then begin
               M := R;
            end; {if L < R}
         end; {if R <= Heapsize}
```

```
         if CellArray[CellArray[M]^.NSort]^.Ndensity <
               CellArray[CellArray[i]^.NSort]^.Ndensity then begin
            Temp := CellArray[i]^.Nsort;
            CellArray[i]^.NSort := CellArray[M]^.NSort;
            CellArray[M]^.NSort := Temp;
            NHeapify(CellArray,M);
         end; {if M > i}
      end; {if L <= Heapsize}
   end; {NHeapify}
{-------------------------------------------------------------------------}
function NHeapExtractMax(var CellArray : CellArrayType) : INTEGER;
{returns cell at top of heap; updates heap}

   begin {HeapExtractMax}
      NHeapExtractMax := CellArray[1]^.NSort;
      CellArray[1]^.NSort := CellArray[HeapSize]^.NSort;
      Dec(HeapSize);
      NHeapify(CellArray,1);
   end; {HeapExtractMax}
{-------------------------------------------------------------------------}
procedure PHeapify(var CellArray: CellArrayType; i : integer);
{Maintains binary heapsort property for i and its children.
 Assumes 'children' of i have heap property.}
   var
      L,
      R,
      M,
      Temp : integer;

   begin {PHeapify}
      L := i SHL 1;           {2i}
      R := (i SHL 1) OR 1;    {2i+1}
      if L <= Heapsize then begin {if I is leaf, exit}
         M := L;
         if R <= Heapsize then begin
            if CellArray[CellArray[L]^.PSort]^.Pdensity >
                  CellArray[CellArray[R]^.PSort]^.Pdensity then begin
               M := R;
            end; {if L < R}
         end; {if R <= Heapsize}
         if CellArray[CellArray[M]^.PSort]^.Pdensity <
               CellArray[CellArray[i]^.PSort]^.Pdensity then begin
            Temp := CellArray[i]^.PSort;
            CellArray[i]^.PSort := CellArray[M]^.PSort;
            CellArray[M]^.PSort := Temp;
            PHeapify(CellArray,M);
         end; {if M > i}
      end; {if L <= Heapsize}
   end; {PHeapify}
{-------------------------------------------------------------------------}
function PHeapExtractMax(var CellArray : CellArrayType) : INTEGER;
{returns cell at top of heap; updates heap}

   begin {HeapExtractMax}
      PHeapExtractMax := CellArray[1]^.PSort;
      CellArray[1]^.PSort := CellArray[HeapSize]^.PSort;
      Dec(HeapSize);
      PHeapify(CellArray,1);
   end; {HeapExtractMax}

{-------------------------------------------------------------------------}
{ MAIN PROGRAM                                                            }
{-------------------------------------------------------------------------}

begin
   {Reseed Random Number Generator}
   Randomize;
```

```
Clrscr;
Writeln('Enter choice of:');
Writeln('TripWire Field     Motion Model       FA Level');
Writeln('  [S]poke        [N]o Velocity State   [Z]ero = 0.00');
Writeln('  [R]andom       [V]elocity State      [L]ow  = 0.01');
Writeln('  [G]rid                               [H]igh = 0.10');
Writeln('  [P]roximity                          0.0[0] .. 0.10[A]');
readln(S);
S[1] := UpCase(S[1]);
S[2] := UpCase(S[2]);
S[3] := UpCase(S[3]);
CH := 'N';
case S[1] of
     'S' : begin
              TWFName := TWFNameS;
              Writeln('Enter number of spokes per sextant:');
              Writeln('  [1] through 1[0]');
              readln(CH);
              TWFName[17] := CH;
           end; {'S'}
     'R' : TWFName := TWFNameR;
     'G' : TWFName := TWFNameG;
     'P' : begin
              TWFName := TWFNameP;
              Writeln('Enter relative proximity of sources:');
              Writeln('  [S]=0.6 .. [Z]=0.95 ');
              Writeln('  [0]=1.0 .. [9]=1.45 ');
              Writeln('  [A]=1.5 .. [C]=1.6 ');
              readln(CH);
              TWFName[17] := CH;
           end; {'P'}
     else begin
        Writeln('Invalid Response');
        Halt;
     end; {else}
end; {case}
if S[2] = 'N' then
   for C := 0 to 6 do
      for TC := 0 to 6 do
         {make all transition matrix entries independent of current
          velocity state}
         TMotion[C,TC] := TMotion[7,TC];
FalseProbAssumed := 0.01;
case S[3] of
     'Z' : begin
              FalseProb := 0.0;
              FalseProbAssumed := 0.0;
           end; {'Z'}
     'L' : FalseProb := 0.01;
     'H' : begin
              FalseProb := 0.1;
              FalseProbAssumed := 0.1;
           end; {'Z'}
     '0' : FalseProb := 0.0;
     '1' : FalseProb := 0.01;
     '2' : FalseProb := 0.02;
     '3' : FalseProb := 0.03;
     '4' : FalseProb := 0.04;
     '5' : FalseProb := 0.05;
     '6' : FalseProb := 0.06;
     '7' : FalseProb := 0.07;
     '8' : FalseProb := 0.08;
     '9' : FalseProb := 0.09;
     'A' : FalseProb := 0.1;
     else begin
        Writeln('Invalid Response');
        Halt;
```

```
     end; {else}
end; {case}
if CH <> 'N' then begin     {SimFile stores statistical information}
   if S[1] = 'S' then
      SimFName := '\tp\tripwire\simoutD'+CH+'.'+S
   else
      SimFName := '\tp\tripwire\simoutP'+CH+'.'+S;
end {if}
else
   SimFName := '\tp\tripwire\simout.'+S;

Clrscr;
Writeln('Initializing detection probabilities...please wait...');
InitStatus := 0;

{Load TW's from file}
Assign(TWFile,TWFName);
Reset(TWFile);
{TWFile has one line for each tripwire in the following format:
    SourceX    SourceY    ReceiverX    ReceiverY    99    }
NumTW := 1;
while (NumTW <= MaxTW) and (not EOF(TWFile)) do begin
   NEW(TWArray[NumTW]);
   with TWArray[NumTW]^ do begin
      readln(TWFile,X1,Y1,X2,Y2,Check);
   end; {with}
   if Check = 99 then
      Inc(NumTW);
end; {while}
Dec(NumTW);
Close(TWFile);

{Open output file}
Assign(SimFile,SimFName);
Rewrite(SimFile);
writeln(SimFile,
'Negative Information Tracker  Positive Information Tracker');
writeln(SimFile,
'       MMD       CDF                  MMD       CDF');
Close(SimFile);

{Initialize CellArray}
for C := 1 to Cells do begin
   X := (C-1) mod XCells + 1;
   Y := (C-1) div XCells + 1;
   NEW(CellArray[C]);
   with CellArray[C]^ do begin
      {calculate center of cell}
      if Y mod 2 = 0 then
         CellX := (X-1/2) * 1.7320508*Radius
      else
         CellX :=    X    * 1.7320508*Radius;
      CellY    := (Y-1/3) * 1.5*Radius;
      {assume target distributed uniformly}
      Ndensity := 1.0/Cells;
      Pdensity := 1.0/Cells;
      {initialize other variables}
      NAP := 1.0;
      TWList := nil;
      NSort := C;
      PSort := C;
   end; {with}
end; {for}

{Initialize Target Position and Velocity}
{assume target distributed uniformly}
TargetC := Trunc(1 + Random*Cells);
```

```
LastMove := Trunc(7*Random) mod 7;

{Initialize Negative probabilities}
for TC := 1 to NumTW do begin
   with TWArray[TC]^ do begin
      for C := 1 to Cells do begin
         with CellArray[C]^ do begin
            {ensure forward scattered detection (> 90 degrees)}
            if (X1-CellX)*(X2-CellX) +
               (Y1-CellY)*(Y2-CellY) < 0.0 then begin
               {calculate denominator to avoid divide by zero}
               TWLen := Sqrt(((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1)));
               if TWLen > 0.0 then begin
                  {calculate projection onto tripwire}
                  Proj := ((X2-X1)*(CellX-X1) + (Y2-Y1)*(CellY-Y1))/TWLen;
                  {calculate distance from tripwire}
                  Dist := ((CellX-X1)*(CellX-X1) +
                           (CellY-Y1)*(CellY-Y1) -
                                (Proj*Proj));
                  {account for small negative numbers as a result of
                   floating-point round-off error}
                  if Dist <= 0.0 then Dist := 0.0
                     else Dist := Sqrt(Dist);
                  {calculate width of square with same center as cell,
                   equal area}
                  RadEq := Sqrt(Radius*Radius*(3/8)*Sqrt(3));
                  {calculate proportion of this square within one mile
                   of tripwire}
                  if Dist < 1 - RadEq then
                     {entire square within 1 of TW}
                     Prob := DetectProb
                  else
                     {part of square is outside 1 of TW}
                     Prob := DetectProb*(RadEq + 1 - Dist)/(2*RadEq);
                  if Prob > 0.0 then begin
                     {add this TW to list for this cell}
                     TempPtr := TWList;
                     NEW(TWList);
                     with TWList^ do begin
                        TWNum := TC;
                        AlmProb := 1.0 - (1.0-Prob)*(1.0-FalseProbAssumed);
                        NAP := NAP * (1.0 - AlmProb);
                        Next := TempPtr;
                     end; {with}
                  end {if}
                  else begin
                     NAP := NAP * (1.0 - FalseProbAssumed);
                    {if Prob <= 0.0 then AlmProb := FalseProbAssumed.
                     This number is not assigned to a variable
                     to conserve memory}
                  end; {else}
               end {if}
               else begin
                  NAP := NAP * (1.0 - FalseProbAssumed);
               end; {else}
            end {if}
            else begin
               NAP := NAP * (1.0 - FalseProbAssumed);
            end; {else}
         end; {with}
      end; {for}
      Inc(InitStatus);
      GotoXY(1,2);
      writeln(100*InitStatus/NumTW:5:1,' % ');
   end; {with}
end; {for}
```

```
InitGraphics;
GraphicsOn := TRUE;
SetTextStyle(SmallFont,HorizDir,6);
SetTextJustify(CenterText,TopText);
OutTextXY(160,0,'Negative Information Tracker');
OutTextXY(480,0,'Positive Information Tracker');
{Set graph palette colors}
SetPalette(1,56);  {Dark gray}
SetPalette(2,24);
SetPalette(3,16);  {Dark green}
SetPalette(4,10);
SetPalette(5,34);
SetPalette(6,26);  {in between}
SetPalette(7,58);
SetPalette(8,30);
SetPalette(9,62); {Bright yellow}

{Main operating loop}
SimTime := 0;
while SimTime < StopTime do begin;

   Inc(SimTime);

   {if at end of iterations, re-randomize target}
   if (SimTime mod Iterations) = 1 then begin
      {Initialize Target Position and Velocity}
      TargetC := Trunc(1 + Random*Cells);
      LastMove := Trunc(7*Random) mod 7;
      {Initialize Target Distribution}
      for C := 1 to Cells do begin
         with CellArray[C]^ do begin
            Ndensity := 1.0/Cells;
            Pdensity := 1.0/Cells;
            NSort := C;
            PSort := C;
         end; {with}
      end; {for}
   end; {if}

   {Move real target; for generation of positive information}
   PTotalProb := Random;
   TC := 0;
   while TC < 7 do begin
      {Move target conditional on velocity state}
      PTotalProb := PTotalProb - TMotion[LastMove,TC];
      if PTotalProb < 0.0 then begin
         {this is selected movement}
         if ((TargetC-1) div XCells) mod 2 = 0 then begin    {odd row}
            case TC of
               1 : if TargetC > XCells then
                        TargetC := TargetC - XCells;
               2 : if (TargetC > XCells) and
                           (TargetC mod XCells <> 0) then
                        TargetC := TargetC - XCells + 1;
               3 : if TargetC mod XCells <> 1 then
                        TargetC := TargetC - 1;
               4 : if TargetC mod XCells <> 0 then
                        TargetC := TargetC + 1;
               5 : if TargetC <= Cells - XCells then
                        TargetC := TargetC + XCells;
               6 : if (TargetC <= Cells - XCells) and
                        (TargetC mod XCells <> 0) then
                        TargetC := TargetC + XCells + 1;
            end; {case}
         end else begin                                      {even row}
            case TC of
               1 : if TargetC mod XCells <> 1 then
```

```
                    TargetC := TargetC - XCells - 1;
               2 : TargetC := TargetC - XCells;
               3 : if TargetC mod XCells <> 1 then
                    TargetC := TargetC - 1;
               4 : if TargetC mod XCells <> 0 then
                    TargetC := TargetC + 1;
               5 : if (TargetC <= Cells - XCells) and
                       (TargetC mod XCells <> 1) then
                    TargetC := TargetC + XCells - 1;
               6 : if TargetC <= Cells - XCells then
                    TargetC := TargetC + XCells;
            end; {case}
         end; {if}
         LastMove := TC;
         {force exit from loops}
         TC := 7;
      end; {if}
      Inc(TC);
   end; {while}

   {Update Ndensity and Pdensity Array for target movement}
   {First loop will create Temp as new distribution; second loop
    reassigns Temp to NDensity}
   for C := 1 to Cells do begin
      if ((C-1) div XCells) mod 2 = 0 then begin   {odd row}
         if C > XCells then
            AdjCells[1] := C - XCells
         else AdjCells[1] := C;
         if (C > XCells) and (C mod XCells <> 0) then
            AdjCells[2] := C - XCells + 1
         else AdjCells[2] := C;
         if C mod XCells <> 1 then
            AdjCells[3] := C - 1
         else AdjCells[3] := C;
         if C mod XCells <> 0 then
            AdjCells[4] := C + 1
         else AdjCells[4] := C;
         if C <= Cells - XCells then
            AdjCells[5] := C + XCells
         else AdjCells[5] := C;
         if (C <= Cells - XCells) and (C mod XCells <> 0) then
            AdjCells[6] := C + XCells + 1
         else AdjCells[6] := C;
      end {if}
      else begin                                   {even row}
         if C mod XCells <> 1 then
            AdjCells[1] := C - XCells - 1
         else AdjCells[1] := C;
         AdjCells[2] := C - XCells;
         if C mod XCells <> 1 then
            AdjCells[3] := C - 1
         else AdjCells[3] := C;
         if C mod XCells <> 0 then
            AdjCells[4] := C + 1
         else AdjCells[4] := C;
         if (C <= Cells - XCells) and (C mod XCells <> 1) then
            AdjCells[5] := C + XCells - 1
         else AdjCells[5] := C;
         if C <= Cells - XCells then
            AdjCells[6] := C + XCells
         else AdjCells[6] := C;
      end; {else}
      CellArray[C]^.Temp := CellArray[C]^.Ndensity*TMotion[7,0];
      for TC := 1 to 6 do
         CellArray[C]^.Temp := CellArray[C]^.Temp +
               CellArray[AdjCells[TC]]^.Ndensity*TMotion[7,7-TC];
   end; {for}
```

```
for C := 1 to Cells do begin
   with CellArray[C]^ do begin
      Ndensity := Temp;
   end; {with}
end; {for}

{First loop will create Temp as new distribution; second loop
 reassigns Temp to PDensity}
for C := 1 to Cells do begin
   if ((C-1) div XCells) mod 2 = 0 then begin    {odd row}
      if C > XCells then
         AdjCells[1] := C - XCells else AdjCells[1] := C;
      if (C > XCells) and (C mod XCells <> 0) then
         AdjCells[2] := C - XCells + 1 else AdjCells[2] := C;
      if C mod XCells <> 1 then
         AdjCells[3] := C - 1 else AdjCells[3] := C;
      if C mod XCells <> 0 then
         AdjCells[4] := C + 1 else AdjCells[4] := C;
      if C <= Cells - XCells then
         AdjCells[5] := C + XCells else AdjCells[5] := C;
      if (C <= Cells - XCells) and (C mod XCells <> 0) then
         AdjCells[6] := C + XCells + 1 else AdjCells[6] := C;
   end {if}
   else begin                                    {even row}
      if C mod XCells <> 1 then
         AdjCells[1] := C - XCells - 1 else AdjCells[1] := C;
      AdjCells[2] := C - XCells;
      if C mod XCells <> 1 then
         AdjCells[3] := C - 1 else AdjCells[3] := C;
      if C mod XCells <> 0 then
         AdjCells[4] := C + 1 else AdjCells[4] := C;
      if (C <= Cells - XCells) and (C mod XCells <> 1) then
         AdjCells[5] := C + XCells - 1 else AdjCells[5] := C;
      if C <= Cells - XCells then
         AdjCells[6] := C + XCells else AdjCells[6] := C;
   end; {if}
   CellArray[C]^.Temp := CellArray[C]^.Pdensity*TMotion[7,0];
   for TC := 1 to 6 do
      CellArray[C]^.Temp := CellArray[C]^.Temp +
           CellArray[AdjCells[TC]]^.Pdensity*TMotion[7,7-TC];
end; {for}
for C := 1 to Cells do begin
   with CellArray[C]^ do begin
      Pdensity := Temp;
   end; {with}
end; {for}

{Incorporate positive and negative information}
TempPtr := CellArray[TargetC]^.TWList;
{TC will step *down* through all TW's.  The linked list TWList is
 sorted downward, and will advance whenever TC passes each link.
 When TC = TW in link, then both false alarm or real detection are
 possible with that TW; otherwise only false alarm will trigger.}
TC := NumTW;
{Assign NextMatch to first tripwire in actual target cell's list}
if TempPtr = nil then
   NextMatch := 0
else
   NextMatch := TempPtr^.TWNum;

while TC > 0 do begin
   {Assign NextMatch to next tripwire in actual target cell's list}
   if TC < NextMatch then begin
      TempPtr := TempPtr^.Next;
      if TempPtr = nil then
         NextMatch := 0
      else
```

```
            NextMatch := TempPtr^.TWNum;
        end; {if}

        if TC = NextMatch then
            {Alarm if either actual or false detection}
            Alarmed := Random < 1.0 - (1.0-TempPtr^.AlmProb)
                                    *(1.0-FalseProb)/(1.0-FalseProbAssumed)
        else
            {Alarm if false detection only}
            Alarmed := Random < FalseProb;
            {this is value of AlmProb for remote cell-tripwire combinations}
        {Aij = prob tw j alarms given target in cell i
                = AlmProb if cell near tripwire
                = FalseProb otherwise}
        {Pi' = Pi * Prod(j alarmed,Aij)}
        {Ni' = Ni * Prod(j alarmed,Aij/(1-Aij))
          TC = j, C = i}
        if Alarmed then begin
        {iterate through all cells}
        {increase N/PDensity in cells associated with detecting TW}
            for C := 1 to Cells do begin
                {find tripwires near this cell}
                TempPtr2 := CellArray[C]^.TWList;
                Searching := TRUE;
                {loop until detecting tripwire located or determined not
                 near this cell}
                while (TempPtr2 <> nil) and Searching do begin
                    if TempPtr2^.TWNum = TC then begin
                        {detecting TW found in this cell's list}
                        with CellArray[C]^ do begin
                            NDensity := NDensity*TempPtr2^.AlmProb
                                        / (1.0 - TempPtr2^.AlmProb);
                                        {denominator counteracts later
                                        multiplication by NAP}
                            PDensity := PDensity*TempPtr2^.AlmProb;
                        end; {with}
                        Searching := FALSE;
                    end; {if}
                    TempPtr2 := TempPtr2^.Next;
                end; {while}
                {loop exits if TempPtr2 = nil (end of list) or
                               Searching = FALSE (TW found) or both}
                if Searching then begin
                    {detecting TW not found in this cell's list}
                    with CellArray[C]^ do begin
                        NDensity := NDensity*FalseProbAssumed
                                    /(1.0 - FalseProbAssumed);
                                   {denominator counteracts later
                                    multiplication by NAP}
                        PDensity := PDensity*FalseProbAssumed;
                    end; {with}
                end; {if}
            end; {for}
        end; {if}
        {if not Alarmed, later multiplication by NAP will account for this}
        {advance to next TW}
        Dec(TC);
    end; {while}

    NTotalProb := 0.0;
    PTotalProb := 0.0;
    for C := 1 to Cells do begin
        with CellArray[C]^ do begin
            {Incorporate negative information}
            Ndensity := Ndensity * NAP;
            {Calculate sums to normalize}
            NTotalProb := NTotalProb + Ndensity;
```

```
      PTotalProb := PTotalProb + Pdensity;
   end; {with}
end; {for}
NMMD := 0.0;
PMMD := 0.0;
for C := 1 to Cells do begin
   with CellArray[C]^ do begin
      {Normalize}
      Ndensity := Ndensity / NTotalProb;
      Pdensity := Pdensity / PTotalProb;
      {Calculate square of missed distance}
      Dist :=  (CellArray[TargetC]^.CellX - CellX)
             *(CellArray[TargetC]^.CellX - CellX)
            + (CellArray[TargetC]^.CellY - CellY)
             *(CellArray[TargetC]^.CellY - CellY);
      NMMD := NMMD + NDensity * Dist;
      PMMD := PMMD + PDensity * Dist;
   end; {with}
end; {for}
NMMD := Sqrt(NMMD);
PMMD := Sqrt(PMMD);

{Sort by Ndensity/Pdensity}
{Uses Binary Heapsort.  Step 1; build heap sorted with < at top.
                        Step 2; de-heap putting < at bottom.    }
{Step 1}
Heapsize := Cells;
for C := Heapsize div 2 downto 1 do begin
   NHeapify(CellArray,C);
   PHeapify(CellArray,C);
end; {for}
{Step 2}
while Heapsize > 1 do begin
   TC := NHeapExtractMax(CellArray);
   CellArray[Heapsize+1]^.NSort := TC;
   Inc(Heapsize);
   TC := PHeapExtractMax(CellArray);
   CellArray[Heapsize+1]^.PSort := TC;
end; {while}

{Draw Current Target Distribution}
{Toggle graphics on/off to speed display}
if KeyPressed then begin
   CH := ReadKey;
   CH := UpCase(CH);
   if CH = 'G' then begin
      GraphicsOn := not GraphicsOn;
      if not GraphicsOn then
         OutTextXY(320,180,'Graphics Disabled');
   end; {if}
end; {if}

NTotalProb := 1.0;
PTotalProb := 1.0;

if GraphicsOn then begin
   SetFillStyle(SolidFill,Black);
   Bar(0,24,639,359);
end; {if}

for C := 1 to Cells do begin
   if CellArray[C]^.Nsort = TargetC then begin
      NContain := 1.0 - NTotalProb +
           Random*CellArray[TargetC]^.NDensity;
      if NContain <= 0.5 then
         Inc(NIn50AOU);
   end; {if}
```

```
if CellArray[C]^.Psort = TargetC then begin
   PContain := 1.0 - PTotalProb +
        Random*CellArray[TargetC]^.PDensity;
   if PContain <= 0.5 then
      Inc(PIn50AOU);
end; {if}

if GraphicsOn then begin
   with CellArray[CellArray[C]^.Nsort]^ do begin
      if NTotalProb > 0.9460 then Color := 9 else {1/3 s.d.}
      if NTotalProb > 0.8007 then Color := 8 else {2/3 s.d.}
      if NTotalProb > 0.6065 then Color := 7 else {1 s.d.}
      if NTotalProb > 0.4111 then Color := 6 else {4/3 s.d.}
      if NTotalProb > 0.2494 then Color := 5 else {5/3 s.d.}
      if NTotalProb > 0.1353 then Color := 4 else {2 s.d.}
      if NTotalProb > 0.0657 then Color := 3 else {7/3 s.d.}
      if NTotalProb > 0.0286 then Color := 2 else {8/3 s.d.}
      if NTotalProb > 0.0111 then Color := 1 {3 s.d.}
                             else Color := Black;
      if (CellArray[C]^.NSort = TargetC) then Color := WHITE;
      {Update containment region}
      NTotalProb := NTotalProb - Ndensity;
      if Color <> Black then begin
         SetColor(Black);
         SetFillStyle(SolidFill,Color);
         PolyPoints[1]  := 4 +
                           Round(CellX/Radius*Scale);
         PolyPoints[2]  := 24 +
                           Round(CellY/Radius*Scale -
                           Scale);
         PolyPoints[3]  := 4 +
                           Round(CellX/Radius*Scale +
                           Scale*0.8660254);
         PolyPoints[4]  := 24 +
                           Round(CellY/Radius*Scale -
                           Scale*0.5);
         PolyPoints[5]  := 4 +
                           Round(CellX/Radius*Scale +
                           Scale*0.8660254);
         PolyPoints[6]  := 24 +
                           Round(CellY/Radius*Scale +
                           Scale*0.5);
         PolyPoints[7]  := 4 +
                           Round(CellX/Radius*Scale);
         PolyPoints[8]  := 24 +
                           Round(CellY/Radius*Scale +
                           Scale);
         PolyPoints[9]  := 4 +
                           Round(CellX/Radius*Scale -
                           Scale*0.8660254);
         PolyPoints[10] := 24 +
                           Round(CellY/Radius*Scale +
                           Scale*0.5);
         PolyPoints[11] := 4 +
                           Round(CellX/Radius*Scale -
                           Scale*0.8660254);
         PolyPoints[12] := 24 +
                           Round(CellY/Radius*Scale -
                           Scale*0.5);
         FillPoly(6,PolyPoints)
      end; {if}
   end; {with}
   with CellArray[CellArray[C]^.Psort]^ do begin
      if PTotalProb > 0.9460 then Color := 9 else {1/3 s.d.}
      if PTotalProb > 0.8007 then Color := 8 else {2/3 s.d.}
      if PTotalProb > 0.6065 then Color := 7 else {1 s.d.}
      if PTotalProb > 0.4111 then Color := 6 else {4/3 s.d.}
```

```
            if PTotalProb > 0.2494 then Color := 5 else {5/3 s.d.}
            if PTotalProb > 0.1353 then Color := 4 else {2 s.d.}
            if PTotalProb > 0.0657 then Color := 3 else {7/3 s.d.}
            if PTotalProb > 0.0286 then Color := 2 else {8/3 s.d.}
            if PTotalProb > 0.0111 then Color := 1 {3 s.d.}
                                   else Color := Black;
            if (CellArray[C]^.PSort = TargetC) then Color := WHITE;
            {Update containment region}
            PTotalProb := PTotalProb - Pdensity;
            if Color <> Black then begin
               SetColor(Black);
               SetFillStyle(SolidFill,Color);
               PolyPoints[1]  := 324 +
                                 Round(CellX/Radius*Scale);
               PolyPoints[2]  := 24 +
                                 Round(CellY/Radius*Scale -
                                 Scale);
               PolyPoints[3]  := 324 +
                                 Round(CellX/Radius*Scale +
                                 Scale*0.8660254);
               PolyPoints[4]  := 24 +
                                 Round(CellY/Radius*Scale -
                                 Scale*0.5);
               PolyPoints[5]  := 324 +
                                 Round(CellX/Radius*Scale +
                                 Scale*0.8660254);
               PolyPoints[6]  := 24 +
                                 Round(CellY/Radius*Scale +
                                 Scale*0.5);
               PolyPoints[7]  := 324 +
                                 Round(CellX/Radius*Scale);
               PolyPoints[8]  := 24 +
                                 Round(CellY/Radius*Scale +
                                 Scale);
               PolyPoints[9]  := 324 +
                                 Round(CellX/Radius*Scale -
                                 Scale*0.8660254);
               PolyPoints[10] := 24 +
                                 Round(CellY/Radius*Scale +
                                 Scale*0.5);
               PolyPoints[11] := 324 +
                                 Round(CellX/Radius*Scale -
                                 Scale*0.8660254);
               PolyPoints[12] := 24 +
                                 Round(CellY/Radius*Scale -
                                 Scale*0.5);
               FillPoly(6,PolyPoints);
            end; {if}
         end; {with}
      end {if}
      else begin
         NTotalProb := NTotalProb -
                       CellArray[CellArray[C]^.Nsort]^.Ndensity;
         PTotalProb := PTotalProb -
                       CellArray[CellArray[C]^.Psort]^.Pdensity;
      end; {else}
   end; {for}

   SetColor(White);
   SetFillStyle(SolidFill,Black);
   Bar(0,360,639,479);

   Str(NMMD :4:6,S);
   OutTextXY(160,360,'Mean Missed Dist:  '+S);
   Str(PMMD :4:6,S);
   OutTextXY(480,360,'Mean Missed Dist:  '+S);
   Str(NContain :4:6,S);
```

```
      OutTextXY(160,380,'AOU to include T:  '+S);
      Str(PContain :4:6,S);
      OutTextXY(480,380,'AOU to include T:  '+S);
      Str(NIn50AOU/SimTime :4:6,S);
      OutTextXY(160,400,'Prop. in 50% AOU:  '+S);
      Str(PIn50AOU/Simtime :4:6,S);
      OutTextXY(480,400,'Prop. in 50% AOU:  '+S);
      Str(SimTime :6,S);
      OutTextXY(320,440,'Number of iterations:  '+S);

      Append(SimFile);
      writeln(SimFile,NMMD :10:4,
                      NContain :10:4,
                      PMMD :20:4,
                      PContain :10:4);
      Close(SimFile);
   end; {while}

   CloseGraph;
end.
```

LIST OF REFERENCES

Borland International, Inc., *Turbo Pascal Version 7.0*, 1992.

Loane, E. P., *Acoustic Forward Scattering: Surveillance Capabilities and Issues*, Report to Johns Hopkins University Applied Physics Laboratory, September 4, 1992.

Loane, E. P., *Model for Acoustic Forward Scatter Detection*, Memorandum to Johns Hopkins University Applied Physics Laboratory, December 16, 1993.

Loane, E. P., *Surveillance Evaluation Methodology*, Memorandum to Johns Hopkins University Applied Physics Laboratory, September 1, 1993.

Ross, S. M., *Introduction to Probability Models*, Fifth Edition, Harcourt Brace & Company, 1993.

Stone, L. D. and Kratzke, T. M., *Comparison of Linear and NonLinear Trackers*, Report to Naval Research Laboratory, December 16, 1991.

Wagner, D. H., *Naval Tactical Decision Aids*, Lecture Notes, U.S. Naval Postgraduate School, June 1989.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3.	Department of Operations Research Attn: Professor Alan Washburn, Code OR/Ws Naval Postgraduate School Monterey, California 93943-5000	1
4.	Department of Operations Research Attn: Professor Robert Dell, Code OR/De Naval Postgraduate School Monterey, California 93943-5000	1
5.	The Johns Hopkins University Applied Physics Laboratory Attn: L. F. Rogers (7-338) Johns Hopkins Road Laurel, Maryland 20723-6099	1
6.	EPL Analysis, Inc. Attn: E. P. Loane 2919-A Olney-Sandy Spring Road Olney, Maryland 20832	1
7.	Metron, Inc. 11911 Freedom Drive, Suite 800 Reston, Virginia 22090	1
8.	Daniel H. Wagner, Associates Station Square One Paoli, Pennsylvania 19301	1

9. Naval Undersea Warfare Center 1
Detachment New London
New London, Connecticut 06320

10. Naval Command Control and Ocean Surveillance Center 1
San Diego, California 92152-7383

11. LT Daniel B. Widdis, USN 1
85 Woodbridge Drive
Colorado Springs, Colorado 80906-4470